Depth maps estimation and use for 3DTV

Gael Sourimant

▶ **To cite this version:**

Gael Sourimant. Depth maps estimation and use for 3DTV. [Technical Report] RT-0379, INRIA. 2010, pp.69. inria-00458221

HAL Id: inria-00458221
https://hal.inria.fr/inria-00458221

Submitted on 19 Feb 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Depth maps estimation and use for 3DTV

Sourimant Gaël

N° 0379

February 2010

Rapport technique

ISSN 0249-0803 ISRN INRIA/RT--0379--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

Depth maps estimation and use for 3DTV

Sourimant Gaël

Thème : Perception, cognition, interaction
Équipes-Projets Temics

Rapport technique n° 0379 — February 2010 — 69 pages

Abstract: We describe in this document several depth maps estimation methods, in different video contexts. For standard (monocular) videos of fixed scene and moving camera, we present a technique to extract both the 3D structure of the scene and the camera poses over time. These information are exploited to generate dense depth maps for each image of the video, through optical flow estimation algorithms. We also present a depth maps extraction method for multi-view sequences aiming at generating MVD content for 3DTV. These works are compared to existing approaches used at writing time in the 3DV group of MPEG for normalization purposes. Finally, we demonstrate how such depth maps can be exploited to perform relief auto-stereoscopic rendering, in a dynamic and interactive way, without sacrifying the real-time computation constraint.

Key-words: Depth maps, disparity maps, Structure from Motion, Multi-view videos, 3D Videos, 3DTV, Normalization, Auto-stereoscopy, GPGPU, Optical flow

This work has been supported by French national project *Futurim@ges*

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Estimation et utilisation de cartes de profondeur en 3DTV

Résumé : Nous décrivons dans ce document plusieurs méthodes d'estimation de cartes de profondeur, dans des contextes video différents. Pour des vidéos classiques (monoculaires) de scènes fixes avec caméra en mouvement, nous présentons une technique d'estimation de la structure 3D et des poses de la caméra au cours du temps, exploitée pour générer des cartes de profondeur denses pour chaque image avec des algorithmes d'estimation du flot optique. Nous présentons également une méthode d'extraction de cartes de profondeur pour des séquences multi-vues en vue de la génération de contenus MVD pour la TV 3D. Ces travaux sont comparés au approches existantes en cours dans le groupe 3DV de MPEG pour la normalisation. Nous démontrons enfin que de telles cartes peuvent être exploités pour effectuer un rendu en relief auto-stéréóoscopique, de façon dynamique et interractive tout en restant dans les contraintes du calcul temps-réel.

Mots-clés : Cartes de profondeur, cartes de disparité, Structure from Motion, Vidéos multi-vues, Vidéos 3D, 3DTV, Normalisation, Auto-stéréoscopie, GPGPU, Flot optique

Contents

List of Figures

Foreword

1 - The Futurim@ges project

These works have been funded by the French project Futurim@ges, during years 2008 and 2009. The goal of the project itself was to study the future TV video formats that could be introduced before the next five years (so around 2015 by the time the project has been closed). Immersion seems to be a key point of this evolution, and studies have been conducted in three different axes:

- HDTV in the high frequency and progressive mode (1080p50)
- HDR (High Dynamic Range) images with higher contrasts (light power) and extended colorimetric spaces
- 3DTV as the relief visualization without glasses

The INRIA has been involved in this project to work on the 3DTV field, especially on depths maps extraction from video content, and study of 3D representations and their impact on the coding and compression chain.

2 - Purpose of this document

The purpose of this document is to summarize works that have been done in depth maps estimation and uses within the TEMICS team at the INRIA Lab for the past two years. It is not intended to be a course on Computer Vision of Computer Graphics, so the reader should be quite familiar with these fields. Some basic notes are sometimes reminded however.

3 - Required notions

More precisely, the reader should be familiar with the following concepts :

- Computer Vision
  - The principle of depth maps
  - The pinhole camera model
  - Projective geometry
  - Stereo Matching
  - *Further Reading*:
    - Multiple View Geometry ..... [HZ04]
    - Visual 3D Modeling from Images ..... [Pol04]
- Computer Graphics
  - The OpenGL pipeline
  - Shader programming
  - Off-screen rendering
  - *Further Reading*:
    - OpenGL Programming Guide (Red Book) ..... [BSe05]
    - OpenGL Shading Language (Orange Book) ..... [Ros06]
    - OpenGL FrameBuffer Object 101/202 ..... [Jon]

4 - Videos licensing terms

Some of the videos presented in this document, for which depth maps extraction results are illustrated, are not the property of the INRIA. We remind here the attached licensing information.

4.1 - MPEG Videos

4.1.1 - Copyright notice

Individuals and organizations extracting sequences from this archive agree that the sequences and all intellectual property rights therein remain the property of the respective owners listed below. These materials may only be used for the purpose of developing, testing and promulgating technology standards. The respective owners make no warranties with respect to the materials and expressly disclaim any warranties regarding their fitness for any purpose.

4.1.2 - Videos owners

Lovebird1	ETRI (Electronics and Telecommunications Research Institute), MPEG-Korea Forum 161 Gajeong, Yuseong Gu, Daejeon, 305-700, Republic of Korea
Cafe	Gwangju Institute of Science and Technology (GIST) 1 Oryong-dong, Buk-gu, Gwangju, 500-712, Republic of Korea
Newspaper	Gwangju Institute of Science and Technology (GIST) 1 Oryong-dong, Buk-gu, Gwangju, 500-712, Republic of Korea
Book Arrival	Fraunhofer Institut für Nachrichtentechnik, Heinrich-Hertz-Institut, Einsteinufer 37, 10587 Berlin Germany
Balloons	Nagoya University Tanimoto Laboratory, Japan

Table 1. List of the different videos owners presented in this document

4.2 - Home Videos

The videos *Home - New York, Arctic, Kilimanjaro, Coral* are extracted from the movie *Home*, by Yann Arthus-Bertrand.

PART I

Depth maps estimation for monocular videos

5 - Interpolation of existing depth maps

Depth maps estimation from monocular videos is extremely challenging in general. In fact this is an inverse problem, where one wants to retrieve 3D information from 2D datasets (namely the video). In the general case, 2D motion within the video is the only cue used to compute the corresponding 3D information, and this motion can be induced either by moving objects in the scene, or by the camera motion itself. So as to simplify this problem, we restricted our study to videos of static scenes with a moving camera. More specifically, we seek here to extract a depth map for each image of the video. We will first see how such maps can be computed using previous works on the subject. Then, we will present our algorithms to compute the scene structure throughout the whole videos.

5.1 - Previous works

Depth maps estimation from such videos has already been dealt with within the TEMICS team, by Galpin [Gal02] and Balter [Bal05]. These works intend to extract a 3D structure from videos in order to compress them efficiently at low bit rates. The global principle of the method is illustrated on Figure 1. First of all, the video is partitioned into GOPS of variable size, depending on the scene content, and the camera pose (position plus orientation) is estimated using a *Structure from Motion* algorithm for each input image. The depth map for the first GOP image (or *keyframe*) is computed, leading to a simple 3D model describing the whole GOP. This model is textured with the corresponding keyframe.
To sum up, this algorithm transforms a group of N images into a single texture plus depth map, associated with N camera poses. This is why such a representation is very efficient in compression terms at low bit rates. For higher bit rates however, the reconstructed quality drops due to 3D artifacts, such as textures stretching.
We can see on Figure 1 that the video is divided into a number of GOPS largely inferior to the number of input images. As such, few depth maps are estimated. As we seek to find a unique depth map for each input image, we first proposed to compute such a 3D representation, and take advantage of it to interpolate the depth maps for the remaining images (*i.e.* for the non keyframe images).

5.2 - Depth maps interpolation

One solution to associate a depth map to each input image would be to consider successive pairs of images in order to estimate the scene depth. However, the inter-frame displacement is way too small in this case to allow such an estimation. We thus exploit the GOP-based representation presented in the previous section to calculate intermediate maps.
We are no longer in the context of sequential video analysis (in which the GOP selection is made) but rather in the reconstruction — or synthesis — case, where the 3D representation is displayed for each estimated camera position. This is illustrated in the lower part of Figure 1. We describe hereafter three different methods to compute such interpolation: a naive solution, one using blended depth maps and a final one requiring the computation of a global 3D model of the scene.

5.2.1 - Naive solution

Reconstructing the original video is done in the following way: for each estimated camera position (*i.e.* for each temporal instant), a 3D model computed using the associated GOP depth map is displayed for the current pose. The 3D engine used for display is based on OpenGL. Each GOP being associated to a unique depth map, the displayed model is the one linked to the GOP the camera position belongs to.

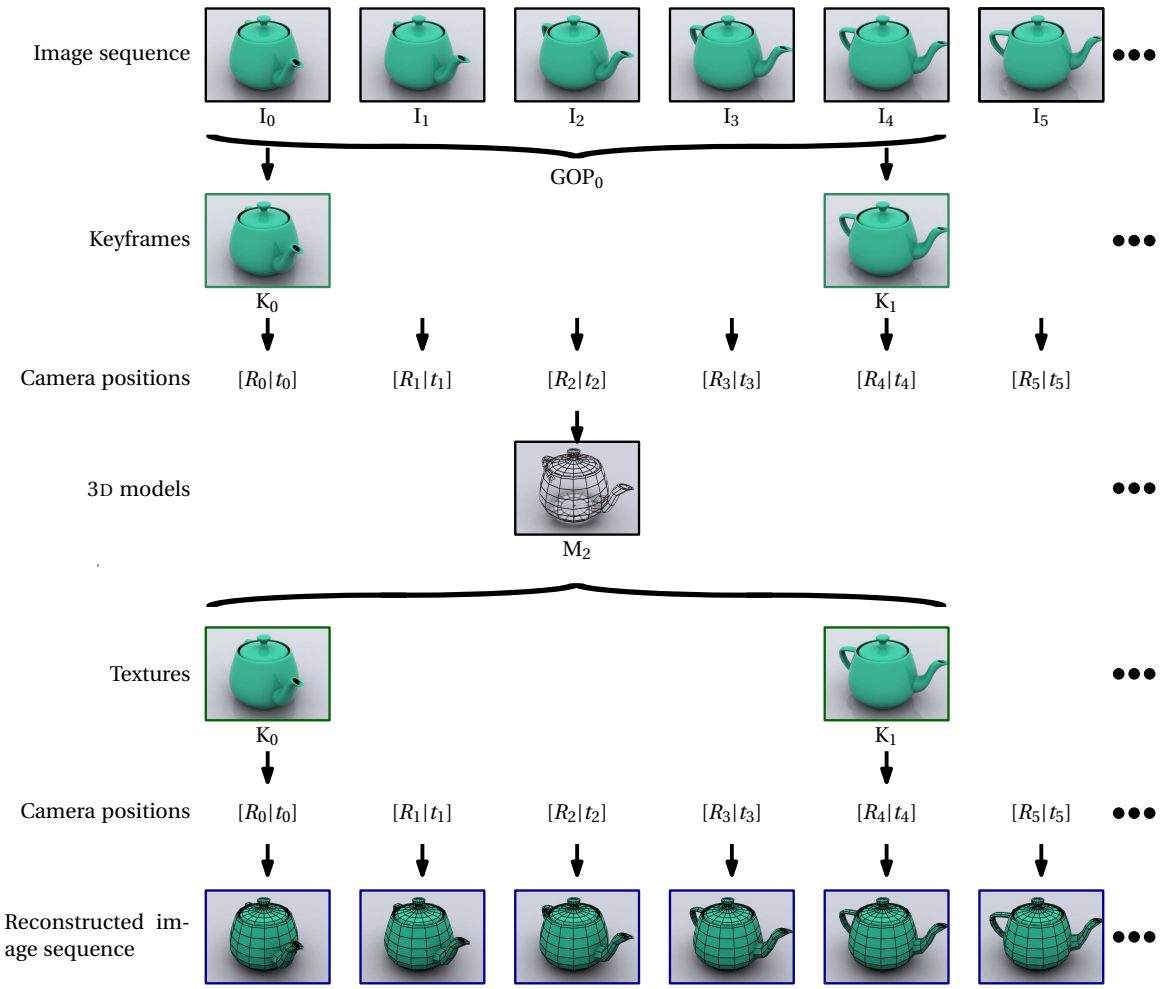


Fig. 1. Global view of Galpin's algorithm

Using the rendering engine, one can thus display the z-buffer of the model instead of its textured version. This z-buffer corresponds to the desired depth map with a proper scaling. An example of such computed maps is given on Figure 2. In that case, the displayed 3D model is a regular mesh for which each node corresponds to a pixel in the depth map.
The major limitation of this approach comes from the use of the meshes associated to each GOP. At GOP transitions, in the classical video reconstruction stage (meaning while displaying textures instead of depth maps), on can observe texture stretching between the last GOP image and the first of the following one (see Figure 3). This effect is caused by two distinct phenomena:

1. The mesh topology does not take into account the depth discontinuities, leading to textures stretching by the end of GOPS.
2. No hard consistency is forced between two successive meshes, and the topological differences thus appear clearly during transitions.

In our case, texture stretching does not occur: we do not aim at reconstructing the images and thus use the original frames instead for each time instant. However, these issues arise also within depth maps. Texture stretching are just replaced by depth discontinuities, which may worsen the future uses of these 2D+Z videos (for instance for auto-stereoscopic display or views interpolation).

(a) Sequence *Saint Sauveur*

(b) Sequence *Cloître*

(c) Sequence *Escalier*

Fig. 2. Naive depth maps estimation by projection

5.2.2 - Depths blending

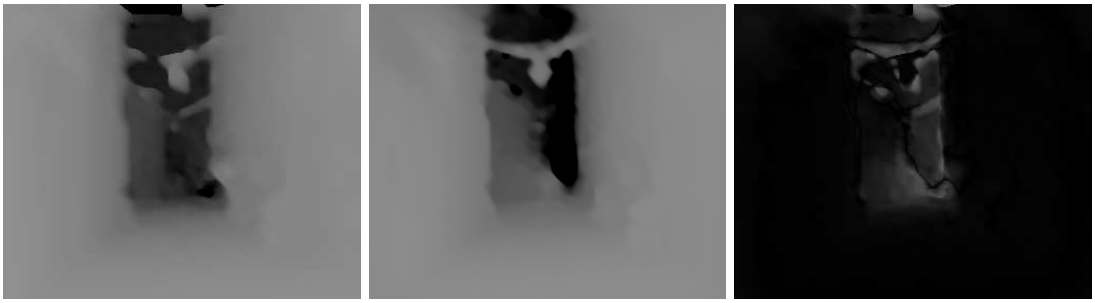
One solution to reduce the artifacts described in the previous section consists in blending the successive meshes over time. Let t_0 be the first image of a GOP, and t_n be the first image of the following GOP. Let t be a frame between t_0 and t_n for which we want to compute a depth map. We compute the depth map Z_t as a linear interpolation between the known depth maps Z_{t_0} and Z_{t_n} associated to t_0 and t_n respectively:

$$Z_t = (1-\alpha)\cdot Z_{t_0} + \alpha \cdot Z_{t_n}, \text{ with } \alpha = (t-t_0)/(t_n - t) \quad (1)$$

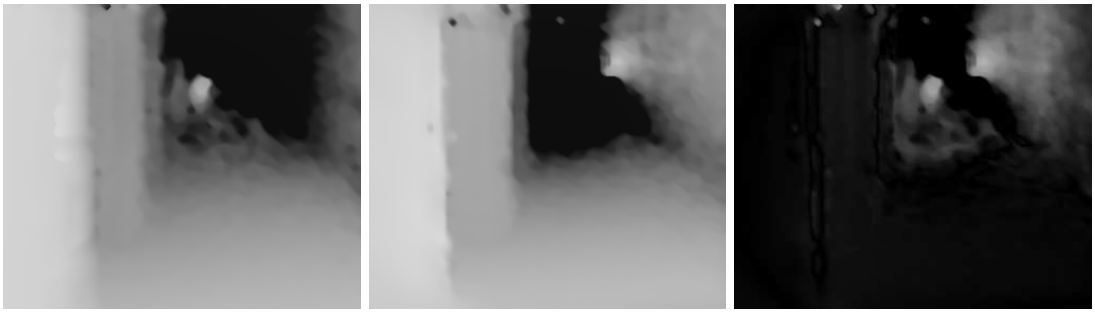
Here α is a time-based linear interpolation, but it could also be a geometric-based interpolation depending on the respective camera positions at times t, t_0 and t_n.
This approach requires a highly accurate 3D registration between the successive meshes. Otherwise, depth inconsistencies appear, especially around meshes boundaries.
This type of interpolation has already been used by [Gal02] in texture space, in order to improve the reconstructed image sequence quality.

5.2.3 - Global model

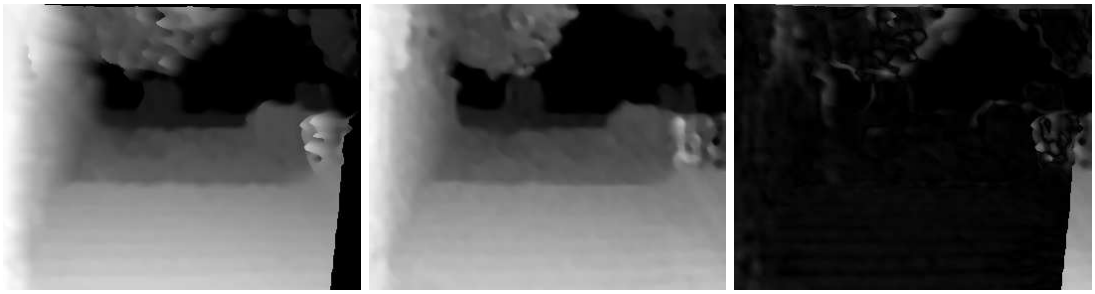
We have seen that GOPS bad transitions are mainly due to topological changes among the successive 3D models. We thus tried to compute a global 3D representation of the scene, which is no longer based on a sequences of meshes, but on a global 3D model which integrates all geometrical information provided by the depth maps of each GOP.
One type of such representation is a volumetric description of the scene using an octree. Using such representation, we discard the artificial connectivity introduced between depth discontinuities (which do not describe the physical state of the scene), and also the stretching occurring at GOPS transitions.



(a) Sequence *Saint Sauveur*



(b) Sequence *Cloître*



(c) Sequence *Escalier*

Fig. 3. Depth maps for successive frames, with difference image

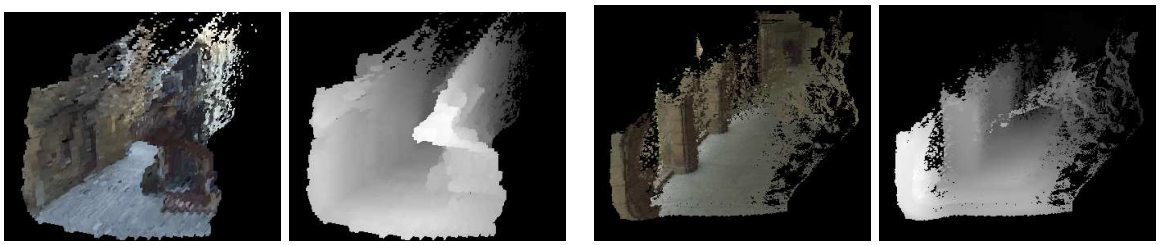


Fig. 4. Virtual view of a volumetric scene representation

This is a direct consequence of the fact that with an octree representation, there is only a single mesh to display.
A rendering of such representation is presented on Figure 4, for a virtual point of view, with both the textured and the z-buffer version. This 3D model is the one displayed all along the video. The octree model is colored using the original images of the video.

(a) Sequence *Saint Sauveur*

(b) Sequence *Cloître*

Fig. 5. Images and associated maps using a global model

Different 2D+Z results are illustrated on Figure 5. Here the depth values have been rescaled for better viewing. The depth maps have been filtered (with a median filter) to smooth the small local discontinuities due to the volumetric representation.

5.2.4 - Discussion

We presented in this section exploratory experiments on depth maps interpolation for 2D+Z videos generation. The presented methods rely heavily on the results of Galpin's algorithm [Gal02]. Unfortunately, this algorithm does not provide accurate enough depth maps. First of all, Galpin's algorithm relies on motion estimator using a deformable 2D mesh, which smooths the resulting vector field, and finally the related depth maps. Such smoothing corrupts the depth values around depth contours; leading to poor interpolated values in these areas.

As a consequence, we present in the following section a new *Structure from Motion* algorithm, based on more recent algorithms of the literature, to replace Galpin's algorithm. We will see how such algorithm can also be exploited to compute accurately depth maps for each input image directly, without any interpolation computation.

6 - Structure from Motion algorithm

We have seen in the previous section that depth maps extraction by interpolation may not be a viable option. This results is mainly due to the fact that camera poses are not extracted accurately enough, leading somewhat to incorrect depth description of the scene. As a consequence, we present here a novel Structure from Motion algorithm (*SfM*), using more recent and reliable works from the literature. This algorithm aims principally at extracting camera poses and scene structure in a single pass. As will be stated below, the structure of the scene will be sparse, meaning that we do not obtain a dense depth representation for each image, but rather a 3D point cloud describing the whole scene. The principal information to extract here is the pose of the camera at each time[1], which will be further used for dense depth maps extraction.

An overview of our method is depicted on Figure 6. Input and/or output data are depicted in straight blue rectangles, while algorithmic blocks are drawn in red rounded rectangles. *SfM* is computed through several passes, and no longer on the flow as in [Gal02]. First, features points are tracked throughout the whole video. Then these points are used to partition the image sequence into GOPS, following Galpin's idea, leading to a pool of keyframes. In a final pass, quasi-euclidean reconstruction (poses and structure) are computed using all keyframes, and remaining camera poses (for non keyframe images) are deduced by interpolation.

6.1 - Feature points extraction and tracking

Our *SfM* algorithm uses feature points as base low level primitives. This is a classical approach, and extraction and tracking of such features are well known problems. The most commonly used feature detectors in the literature are those of Harris & Stephens [HS88], Shi & Tomasi [ST94], SIFT [Low04] or SURF [BTG06]. In this study, we focus SURF features since they are known to outperform other feature types for calibration and 3D reconstruction tasks [BTG06].

6.1.1 - Features extraction

In our algorithm, we use the implementation of SURF features extraction provided by the OpenCV library, within the function `cvExtractSURF()`. An example of extracted feature is presented on Figure 7, for two different frames of a video. Light green dots in the images represent the extracted features with this method.

[1] In this part, we speak indifferently of "poses of the cameras" or "camera poses over time". In fact, assuming that the scene is static and the camera is moving is strictly equivalent to consider a fixed scene with many cameras at different positions for a single time instant.

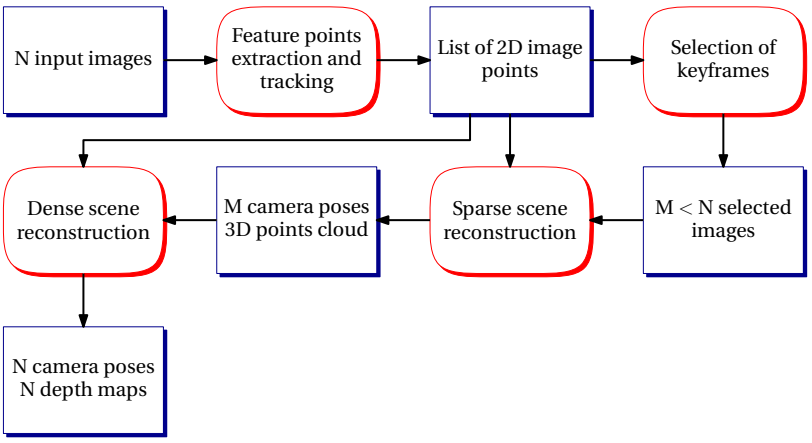


Fig. 6. Overview of our Structure from Motion algorithm

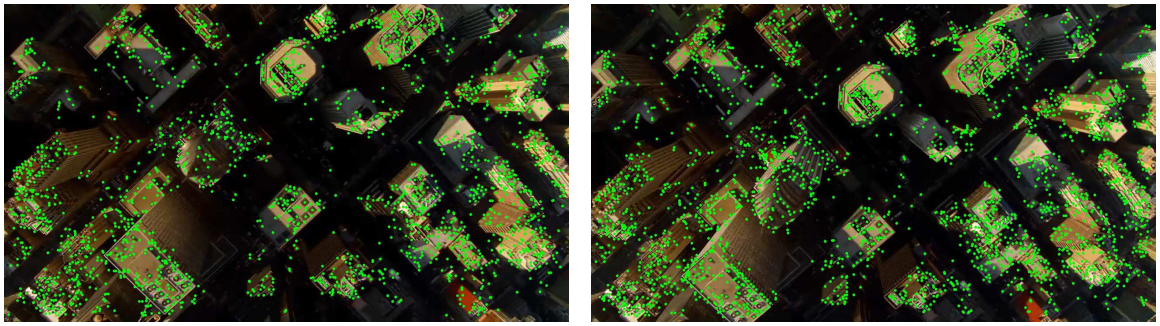


Fig. 7. Extracted SURF features from the sequence *Home - New York*

6.1.2 - Features tracking

Principle In Lowe's original paper [Low04], the features descriptors were used to match the extracted SIFT points across different images. More precisely, if $\mathbf{x}_1$ and $\mathbf{x}_2$ are SIFT features extracted from images I_1 and I_2 respectively, and $\mathbf{d}_1$ and $\mathbf{d}_2$ are their associated descriptors, the point x_1^i is matched to x_2^j if and only if the difference in L^2 norm between their descriptors is sufficiently small, and x_2^j is the closest point to x_1^i among $\mathbf{x}_2$ in terms of that norm:

$$x_1^i \text{ matches } x_2^j \iff x_2^j = \underset{j}{\operatorname{argmin}} \left(\|d_1^i - d_2^j\|^2 \text{ with } \|d_1^i - d_2^j\|^2 < \lambda \right) \tag{2}$$

This principle can also be applied to SIFT features. However, we seek here to find matches across successive images of the very same video. With such data, Lowe's matching method is not the best suited one, since it does not integrates the prior of small inter-frame motion.
A more efficient alternative is to use Tomasi & Kanade feature tracker [TK91, ST94]. It is also implemented in OpenCV within the function `cvCalcOpticalFlowPyrLK()`.

Tracking over time Once features have been detected in the first image, tracking is performed until the last video frame is reached. For each frame, the complete tracking procedure can be split into three steps. First, matches are found between the previous image I_{t-1} and I_t using the method described in the previous paragraph. Then, to ensure matches are robust, an outlier removal pass is applied. This step consists in estimating the epipolar geometry between the two images which is induced by the point matches, by means of the fundamental matrix $\mathbf{F}$:

$$\forall i,\ {x_t^i}^\top \mathbf{F} x_{t-1}^i = \mathbf{0} \tag{3}$$

$\mathbf{F}$ is computed in a robust way using a Least Median of Squares (*LMedS*) approach, given a desired probability that the estimated matrix is correct (typically 0.99). In a very similar way it could be computed within a RANSAC robust procedure [FB81]. Outliers are detected during the robust phase. Such computation can be performed using OpenCV's `cvFindFundametalMat()` function. Finally, points that were lost during the matching process are replaced by new ones. Thus the detection phase is run and only new features that are sufficiently far away from still tracked points are integrated. The minimal distance between old and incoming features is typically about 8 pixels.
Tracking results are depicted on Figure 8, where line segments represent the position over time of the tracked points, for the same images than in Figure 7.

6.2 - Keyframes selection

6.2.1 - Principle

Since we work with videos (as opposed to unordered images), the inter-frame baseline is very narrow, meaning that the camera displacement between two images is very small. As a consequence, it would

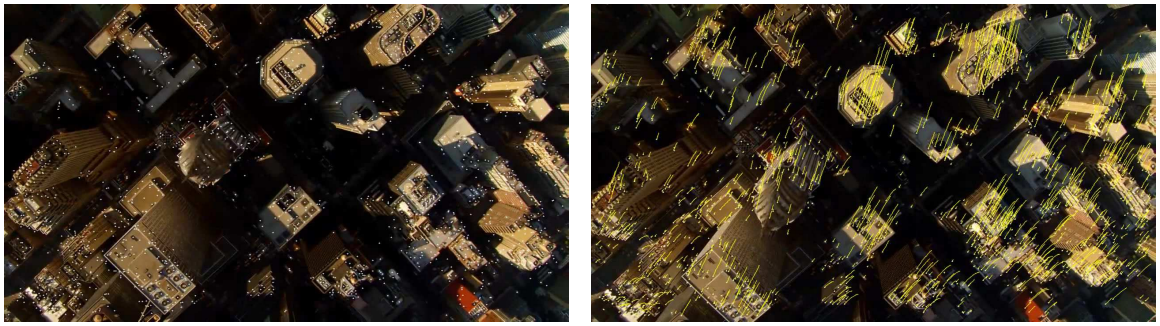


Fig. 8. Tracked SURF features from the sequence *Home - New York*

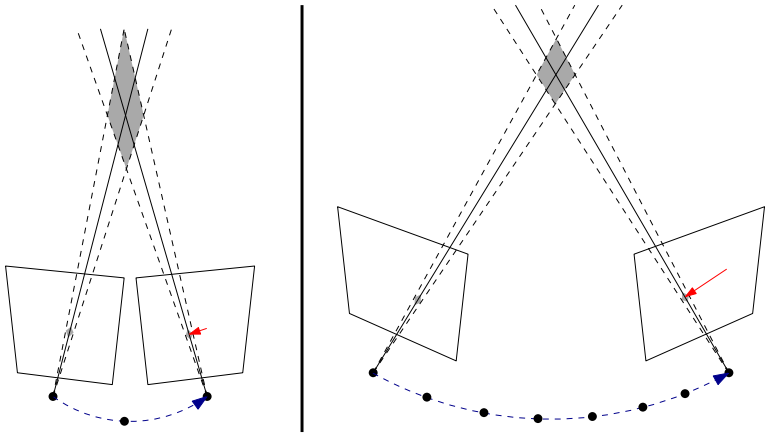


Fig. 9. Relation between baseline and reconstruction error

be a bad idea to try to reconstruct the scene and the camera poses using successive images. In fact, the scene reconstruction error is highly correlated to the relative cameras relative displacement and orientation, which impacts the distance on the 2D point matches. If 2D points measurements are corrupted with (Gaussian) noise, noise influence will decrease with the distance between matches, and as such 3D reconstruction error will also be reduced. This is illustrated on Figure 9. On the left side, the baseline between the two cameras is small, leading to a small 2D displacement (red arrow) and a large potential 3D reconstruction error (gray zone). On the contrary, on the right side the baseline is larger, leading to a small reconstruction error. The dashed lines represent the "error cones" induced by the noise corrupting the 2D points measurements.

It is this need for relevant images to compute the desired reconstruction that imposes to select keyframes from the video. The computation of the camera poses and the scene structure will be performed using point matches across these keyframes. Unfortunately, keyframe selection from videos for reconstruction tasks is not well documented in the literature. To our knowledge, this is mainly due to the fact that such process is highly dependent on the input data, and that writing a generic algorithm to this purpose that works with almost any video "*is still to some extent a black art*" [HZ04].

6.2.2 - Previous works

Several tools to detect which images could be relevant keyframes have however been described. In [Pol04], given a keyframe I_0, the following one I_k maximizes the product between the number of matches and an image-based distance. The proposed distance is the median distance between points $\mathbf{x}_1$ transferred

through an average planar homography $\mathbf{H}$ and the corresponding points $\mathbf{x}_2$ in the target image:

$$dist(I_0, I_k) = \underset{i}{\text{median}} \left(\|\mathbf{H}x_1^i - x_2^i\|^2 \right) \tag{4}$$

A similar procedure is applied in [SSS08], although the data considered here are unordered photographs. Once point matches have been computed, an homography $\mathbf{H}$ is computed between image pairs, through a robust RANSAC procedure, and the percentage of inlier points (*i.e.* for which the homography transfer is a good model) is stored. Then initial reconstruction is performed using the image pair with the lowest number of inliers, provided there are at least 100 matches between them. Indeed, homographic transfer is a very good model for points belonging to images for instance acquired from a pure rotational camera, or for points all belonging to the same plane in space. In both cases, reconstruction is nearly unfeasible, and thus image pairs validating this homographic model should not be considered.
In [Gal02], where video frames are processed on the flow, a more complex procedure is setup to select keyframes. In consists in the combination of several criteria, which are image- and projective-based:

- Lost points during tracking are not replaced, so the percentage of still tracked points should be above a given threshold.
- The mean motion amplitude between tracked points should also be above some given threshold.
- Epipolar geometry is estimated between pairs of images, and only those with a small enough epipolar residual (symmetric distance between points and epipolar lines corresponding with the associated points transfered by the fundamental matrix $\mathbf{F}$) can be selected.

Given a keyframe I_0, the selected next keyframe is the one which validates these three criteria and which is the furthest away from I_0 in terms of frames number.

6.2.3 - Our method

Our approach approach is quite similar to [Gal02]. We combine several criteria in order to select proper keyframes. The main difference is that we compute a *light* prior reconstruction (camera poses and 3D points) to determine whether an image pair is well suited. More details on such reconstruction are detailed in Section 6.3. Given a keyframe I_k, we search for the best next keyframe I_{k+n}, with $n \in \{1; N\}$. First, we remove from the image search space all frames which do not fulfill the following criteria:

- The number of matched points should be large enough.
- The mean 2D motion between matched points should be large enough too.
- The fundamental matrix $\mathbf{F}$ is computed, and the essential matrix $\mathbf{E}$ is then derived by removing the intrinsics influence with user-provided approximate parameters. $\mathbf{E}$ is decomposed into a rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$, corresponding to the relative pose of the cameras, from which we triangulate the matched points to get their 3D position. If a sufficiently large number of points are reconstructed (meaning they are in front of both cameras, see Section 6.3), the reconstruction is considered as viable.

For all frames which passed these three tests, we finally compute a score s_n upon the projective relationships between I_k and I_{k+n}, and the selected keyframe will be the one maximizing s_n:

$$s_n = \epsilon_h^n / \epsilon_e^n \tag{5}$$

ϵ_h^n is the homographic residual. Similarly to [Pol04], we have:

$$\epsilon_h^n = \frac{1}{M} \sum_{i=1}^{M} \left(\|\mathbf{H}x_k^i - x_{k+n}^i\|^2 \right) \text{ with } M = \text{card}(\mathbf{x}_k) \tag{6}$$

In the same way, following [Gal02], ϵ_e^n is the epipolar residual, deduced from $\mathbf{F}$. For two matched points (x_1, x_2) in images I_1 and I_2, $x_1^\top \mathbf{F}$ is the epipolar line l_1 in I_2 corresponding to x_1, while $\mathbf{F}x_2$ is the epipolar line l_2 in I_1 corresponding to x_1. The epipolar residual is then defined as the mean euclidean distance between points and the corresponding epipolar line:

$$\epsilon_e^n = \frac{1}{M} \sum_{i=1}^{M} 0.5 \left(\|x_k^i - \mathbf{F}x_{k+n}^i\|^2 + \|x_{k+n}^i - {x_n^i}^\top \mathbf{F}\|^2 \right) \text{ with } M = \text{card}(\mathbf{x}_k) \tag{7}$$

Once a keyframe has been selected, it is set as the initial frame to search the following keyframe within the video, until the last image is reached.

6.3 - Sparse Structure from Motion

At this point, the information extracted from input images are a list of 2D matched points at each time, and a segmentation of the video into keyframes for which we assume a good reconstruction (3D points and poses) can be derived. Since we do not have any metric information on the viewed scene, our reconstruction will be at best euclidean, and defined up to a scale factor. As such, we first compute the scene reconstruction from the two first keyframes, while fixing this scale. The reconstruction is then upgraded by adding sequentially the remaining keyframes one after the other.

6.3.1 - Initial reconstruction

During this phase, we compute the relative pose of the two keyframes together with the 3D coordinates of the feature points matched across them. It is thus very similar to the last removal criteria described in Section 6.2.3. However, the reconstruction is here computed more precisely. We now explain with more detail this step.

Pose estimation Let I_{k_0} and I_{k_1} be the first two keyframes. Here we derive the pose of these two using only the measured 2D point matches. The reason why we speak of *relative* pose is that we only define the pose for I_{k_1}. Indeed, we initialize the pose $[\mathbf{R}|\mathbf{t}]_{k_0}$ to be at the origin of an arbitrary 3D space, with no orientation component:

$$[\mathbf{R}|\mathbf{t}]_{k_0} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{8}$$

For such camera pair configuration, one speaks of *canonical* cameras. Notice that the camera for image I_{k_0} thus corresponds to the origin of the 3D space in which our reconstruction will be expressed.

To determine $[\mathbf{R}|\mathbf{t}]_{k_1}$, we first compute the fundamental matrix $\mathbf{F}$ in a RANSAC-based robust approach as already stated. This matrix holds the projective relationship relating the camera pair given the matched points. $\mathbf{F}$ is then transformed into an essential matrix $\mathbf{E}$ by removing the (projective) influence the intrinsic parameters held in $\mathbf{K}$:

$$\mathbf{E} = \mathbf{K}^\top \mathbf{F} \mathbf{K} \tag{9}$$

Notice that we do not know precisely these parameters. Instead, we use an approximation provided by the user.

The interesting thing about $\mathbf{E}$ is that it can itself be decomposed into the product of a rotational and a translational components (for more details on the following statements, and associated proofs, see [HZ04]):

$$\mathbf{E} = [\mathbf{t}]_\times \mathbf{R}, \text{ with the skew-symmetric matrix } [\mathbf{t}]_\times = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \tag{10}$$

$\mathbf{R}$ and $\mathbf{t}$ are retrieved using the properties of $\mathbf{E}$. The essential matrix is a 3×3 matrix with rank 2, thus it's third singular value is zero. Moreover, the other two singular values are equal. This come from the product decomposition of $\mathbf{E}$ holding a skew-symmetric matrix. So the first step to find $\mathbf{R}$ and $\mathbf{t}$ is to compute a SVD of $\mathbf{E}$ and then normalize it:

$$\mathbf{E} = \mathbf{U}\mathbf{D}\mathbf{V}^\top \rightsquigarrow \mathbf{U}\,\mathrm{diag}(1,1,0)\,\mathbf{V}^\top \tag{11}$$

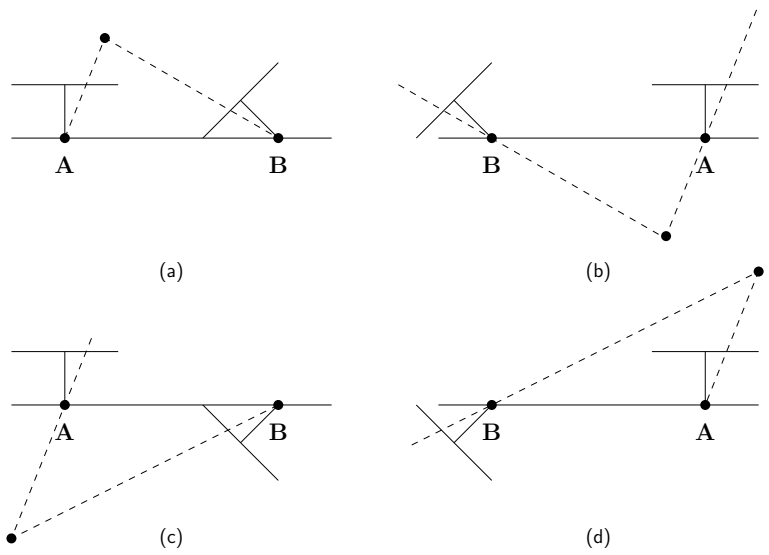


Fig. 10. The four possible decompositions of the essential matrix

Then, from this decomposed essential matrix, there are four possible choices for the camera pose $\mathcal{P}_{k_1} = [\mathbf{R}|\mathbf{t}]$, which are:

$$\begin{array}{rll} \mathcal{P}_{k_1} = & [\mathbf{U}\mathbf{W}\mathbf{V}^\top & | +\mathbf{u}_3] \\ \text{or} & [\mathbf{U}\mathbf{W}\mathbf{V}^\top & | -\mathbf{u}_3] \\ \text{or} & [\mathbf{U}\mathbf{W}^\top\mathbf{V}^\top & | +\mathbf{u}_3] \\ \text{or} & [\mathbf{U}\mathbf{W}^\top\mathbf{V}^\top & | -\mathbf{u}_3] \end{array} \tag{12}$$

$$\text{with } \mathbf{t} = \mathbf{U}[0,0,1]^\top = \mathbf{u}_3 \text{ and } \mathbf{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Notice that here the (unknown) scale is set by enforcing the constraint $\|\mathbf{t}\| = 1$, which is a convenient normalization for the baseline of the two camera matrices.

Geometrically, these four choices (depending on the signs of $\mathbf{R}$ and $\mathbf{t}$) correspond to the four possible camera configurations depicted on Figure 10. Between the left and right sides there is a baseline reversal. Between the top and bottom lines, camera B rotates with a $180°$ angle around the baseline. As one can see, a reconstructed point $\mathbf{X}$ will be in front of both cameras in one of these configurations only (case (a)).

Therefore the last step to determine the choice of $\mathbf{R}$ and $\mathbf{t}$ can be made by triangulating matched points and check whether they are in front of both cameras of not. We now explain how such triangulation can be performed.

Triangulation Here we do not aim at computing precise 3D position of matched points. Such accurate measures are derived once camera poses are estimated (see Section 6.3.2). We describe here a simple linear triangulation method. In each image, we have corresponding measurements $\mathbf{x}_1 \sim \mathcal{P}_1\mathbf{X}$ and $\mathbf{x}_2 \sim \mathcal{P}_2\mathbf{X}$. These equations can be combined to form a linear system of the form $\mathbf{A}\mathbf{X} = \mathbf{0}$. First, the homogeneous scale factor is eliminated by a cross product for each image points, giving three equations of which two are linearly independent. For instance, writing $\mathbf{x} \times (\mathcal{P}\mathbf{X}) = \mathbf{0}$ gives:

$$\begin{array}{rcrcl} x\,(\mathbf{p}^{3\top}\mathbf{X}) & - & (\mathbf{p}^{1\top}\mathbf{X}) & = & 0 \\ y\,(\mathbf{p}^{3\top}\mathbf{X}) & - & (\mathbf{p}^{2\top}\mathbf{X}) & = & 0 \\ x\,(\mathbf{p}^{2\top}\mathbf{X}) & - & y\,(\mathbf{p}^{1\top}\mathbf{X}) & = & 0 \end{array} \tag{13}$$

where $\mathbf{p}^{i\top}$ are the rows of $\mathcal{P}$. An equation of the form $\mathbf{AX} = \mathbf{0}$ can then be composed:

$$\mathbf{A} = \begin{bmatrix} x_1\,\mathbf{p}_1^{3\top} - \mathbf{p}_1^{1\top} \\ y_1\,\mathbf{p}_1^{3\top} - \mathbf{p}_1^{2\top} \\ x_2\,\mathbf{p}_2^{3\top} - \mathbf{p}_2^{1\top} \\ y_2\,\mathbf{p}_2^{3\top} - \mathbf{p}_2^{2\top} \end{bmatrix} \tag{14}$$

This is a redundant set of equations since the solution is determined up to scale. This solution is found using the *Direct Linear Transformation* method (DLT): $\mathbf{X}$ is the unit singular vector corresponding to the smallest singular value of $\mathbf{A}$, which is retrieved using a SVD decomposition.

There are two particular things to notice in this triangulation problem. First, the poses used are those modeling the projection, meaning that the intrinsic parameters are taken into account: $\mathcal{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$. Second, since linear problems are very sensitive to noise, input data are normalized in an isotropic way. For each image, we compute the 3×3 transformation $\mathbf{T}$ which brings the 2D points $\mathbf{x}$ in such a way that the origin of their coordinates is their centroid, and the mean distance to their centroid equals $\sqrt{2}$. The transformation is applied to both matched points and projection matrices, thus providing new input data to the triangulation process, which are less sensitive to noise:

$$\bar{\mathbf{x}} = \mathbf{T}\mathbf{x} \quad \text{and} \quad \bar{\mathcal{P}} = \mathbf{T}\mathcal{P} \tag{15}$$

We also implemented the optimal non-iterative triangulation method presented in [HS97]. It relies on the modification of the 2D points coordinated to minimize the projection error. However, it did not provide more valuable results than the DLT associated to bindle adjustment.

Back to pose selection Testing with a single point to determine if it is in front of both camera is theoretically sufficient to determine which of the four solutions for $[\mathbf{R}|\mathbf{t}]$ has to be picked. However, due to numerical instability and possible errors in points measurements, all points are not always in front of both cameras. As a consequence, we triangulate all points and pick the configuration which maximizes the number of front points. Those that do not fulfill this constraint are removed from consideration.

Refinement by resection At this point, we computed the pose for the first two keyframes, together with and initial scene structure based on matched points. As a final step to this estimation, we refine the pose for I_{k_1} in a non linear way, using the resection principle. Given matches between measured 2D points and computed 3D points, we use the projective relationship $\mathbf{x} \sim \mathcal{P}\mathbf{X}$ to get a finer estimation of the 6 unknown parameters of $\mathcal{P}$ (3 parameters for rotation, and 3 for translation). We use a Levenberg-Marquardt optimization to find the pose that minimizes the projection (euclidean) error between the 2D points and the projected 3D points, using the decomposition of $\mathbf{E}$ as an initialization.
The most important thing here is to correctly model the rotation. If modeled as a 3×3 matrix, the final solution is not constrained (and with a high probability will not) to be a rotation matrix. As a consequence, the rotation parameters we optimize are the imaginary part of the unit quaternion describing the rotation in $\mathbb{R}^3$. Further information on how such modeling can be integrated and derived in projection equations is given in Appendices B and C.

Finally, the whole structure and motion is put in a 2-views *Bundle Adjustment* step (see Section 6.3.2) in order to refine the estimated position of the 3D points.

6.3.2 - Reconstruction upgrade

Once the first two keyframes are added and the scene scale is set, upgrading the reconstruction is dealt with by adding the subsequent keyframes one after the other, to provide a global set of 3D points and camera poses, all expressed in the first camera's coordinate system.
One has to be very careful here, since this phase is highly prone to drift. There is not one good way to do it, so the ideas expressed below are mainly taken as proposals from [HZ04, Pol04]. As in reconstruction initialization, the new camera pose is estimated, so are the unknown 3D points, and finally all of these are refined in a non linear optimization step.

Pose resection We seek here to compute the pose of keyframe image I_{k_n}. First we establish a list of 2D/ 3D matches from the features that have been tracked from $I_{k_{n-1}}$, and for which the 3D position has already been computed, either during initialization or a previous keyframe integration. These matches are input into a non linear resection step to infer the new camera pose, pose which is initialized to the one found for the previous keyframe in order to begin not far from the solution and avoid if possible local minima during optimization.

Points prediction In this step, we try to fill the holes left by our point tracking system, and add new information to our structure. We try to infer the coordinates of all 2D points that have been tracked until $I_{k_{n-1}}$, but have been lost for I_{k_n}.
For these points, the 3D position is already estimated. The resected new camera pose is used to predict the 2D position by projection. From this list of new 2D points $\tilde{\mathbf{x}}$, we only keep those for which the following constraints are preserved:

1. The fundamental matrix $\mathbf{F}$ derived from matched points is used to measure the epipolar residual ϵ_e between the new point in k_n and the former position in k_{n-1}. A predicted point should not violate the epipolar geometry estimated with trusted measures and points from $\tilde{\mathbf{x}}$ with $\epsilon_e > 1$ pixel are discarded.
2. Predicted points should be color-consistent with matched old ones. Thus the distance between points' color is submitted to thresholding to keep only visually similar points. This operation is just a simple and fast way to remove more undesirable points. The key here is not to find all lost points, but rather ensure that predicted ones keep consistent with the ongoing estimation and avoid drift.

Triangulation The next step consists simply in triangulating points which have been matched between k_{n-1} and k_n but for which no 3D information is available. This is the case when these points correspond to features that have replace lost ones near k_{n-1}. In the same spirit that for initialization, only points that can be triangulated in front of both cameras are kept.

Outliers removal This step can be viewed as a local (in terms of keyframes) consistency check. The key idea remains to avoid drift in both poses and structure estimation. Here, the projection error over successive keyframes is used as an error measure. The number M of integrated keyframes is left to the user, but is (obviously) more than two.
For all known 3D points $\mathbf{X}$, we compute the mean projection error over the last M keyframes, using the estimated camera poses. All features with such mean error superior to some threshold λ are completely discarded from the whole estimation. Assuming that an inlier point should be kept at least 95% of the time, and that measurements are corrupted by a zero mean Gaussian noise with a σ standard deviation, we set λ to $5.99\sigma^2$ ([HZ04], Section 4.7). Traditionally, we set σ to one pixel.
One could point out that as the number of estimated keyframes increases, the projection error change for a given feature is only dependent on its new 2D position. However, as will be stated below, both camera poses and 3D points will eventually be corrected afterwards, thus requiring this consistency check several times over the same keyframes.

Pose resection...again This pose resection step is exactly the same that the one performed at the upgrade beginning. It's purpose is to integrate the corrections that have been made in the features pool to get a better camera pose estimation. The only difference is that it is initialized with the previous estimation instead of the pose of the preceding keyframe. Again, the goal here is to (hopefully) compute data while avoiding numerical drifts.

Bundle Adjustment This is the final step of the keyframe integration process. As for outliers removal, its purpose is to integrate estimated data over several previous keyframes. However, we do not discard features here, but correct the points 3D positions and camera poses at the same time, minimizing the projection error. Notice that here 2D points coordinates are considered as trustful data and as such are not modified. This minimization can be seen as a generalization of the resection estimation over several views, where 3D coordinates are also put in the optimization process.

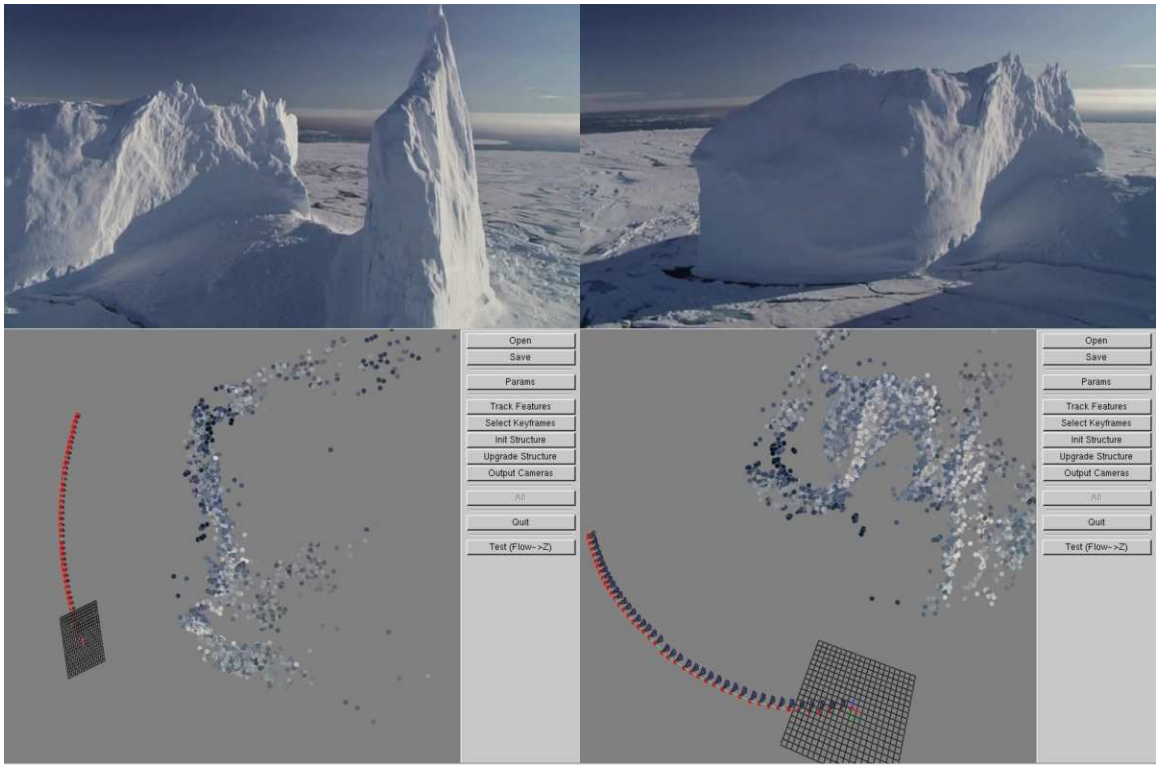


Fig. 11. Sparse SfM for sequence *Home - Arctic*

The minimization itself is performed through a Levenberg-Marquardt loop, but in a specific way. Indeed, minimizing such system requires for instance the inversion of matrices that can be extremely large (more than one million elements), making the optimization process way too time consuming. Fortunately, the system to be minimized is also extremely sparse. This comes from the fact that not all points are viewed from all the cameras. Thus the data matrices can be reorganized to make them less sparse, and more importantly, much smaller [HZ04]. The partial derivatives towards the different projection equations parameters necessary to the LM loop are presented in Appendix C.

In a more technical concern, so as to compute this minimization, we use the *Sparse Bundle Adjustment* library wrote by Lourakis & Argyros [LA09].

6.3.3 - Results

We present in this section some results on sparse reconstruction. On Figures 11, 12 and 13, we present visual results of these reconstructions. On the top row are depicted the first (left) and last (right) images of the video. The bottom row shows two different and virtual points of view the the computed reconstruction. Cameras for the keyframes are showed as blue pyramids, illustrating their position and orientation. The camera centered on the flat grid represents the first camera of the video and the the 3D space origin. The 3D scene points are represented as small colored dots, the color being the one extracted from the corresponding 2D point in the first image it appears.

6.4 - Dense Structure from Motion

Up to this point, we have achieved to build a 3D representation of the input video containing a 3D points cloud modeling the viewed scene, together with an estimated position and orientation of the acquisition camera for a subset of the images. All data in this representation are expressed in a single arbitrary coordinate system, namely the one of the camera for the first image.

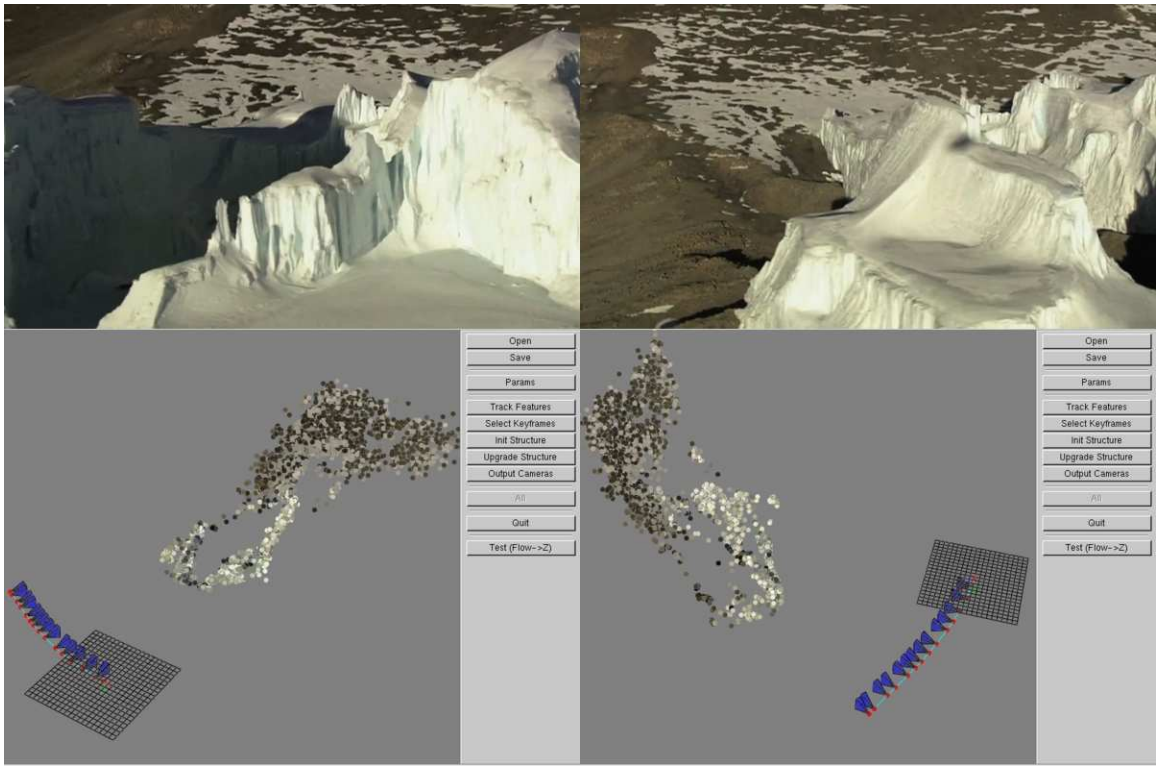


Fig. 12. Sparse SfM for sequence *Home - Kilimanjaro*

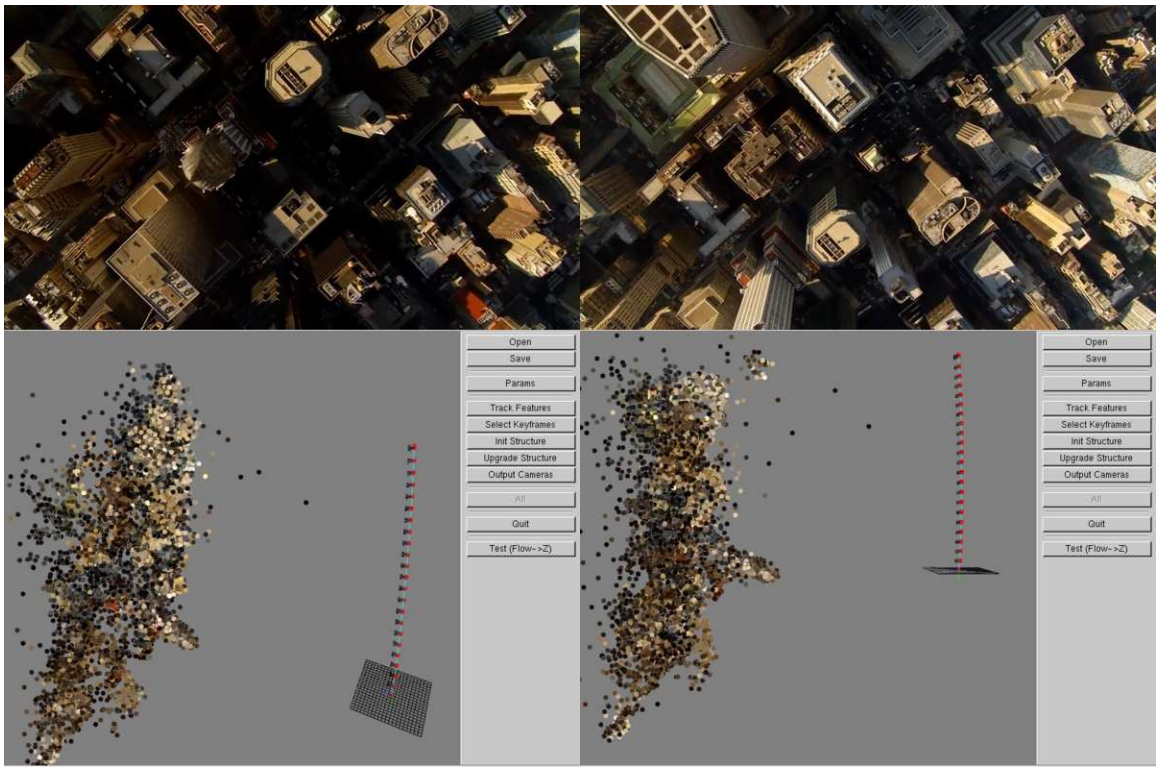


Fig. 13. Sparse SfM for sequence *Home - New York*

However, these works focus on the estimation of depth maps for every single image of the input video. This means that the *sparse* representation we just estimated has to be *densified*. First of all, we have to retrieve estimated for the camera poses not only for the keyframes but for all images. This is motion densification. Secondly, we need to compute a 3D position for each pixel of each image: this is what depth maps are. That corresponds to a structure densification step.
We now highlight how such densification is tackled in our framework.

6.4.1 - Dense motion

Motion estimation for non keyframe images is kind of the easy part of the whole densification step. There are two possible approaches to this problem. The first one is to consider tracked feature points related to estimated 3D points to compute the remaining poses by resection, followed by a non linear refinement step (for instance by running the Bundle Adjustment for all images between two given keyframes). The second one is to assume that cameras for two neighboring keyframes are spatially close enough for cameras in between to be computed by pose interpolation. This is the method we chose.
A warning has to be set here. Though translation interpolation may appear straightforward, rotation interpolation is not. It's quality will be directly related to rotation representation used. For instance — and the same that for pose refinement applies here — interpolating rotation matrices while constraining the result to be still a rotation matrix will be highly challenging. We use instead an angle-vector representation, the vector in consideration being unitary. Ref?

Let's consider here two keyframes, k_1 and k_2. Their translation components are denoted $\mathbf{t}_1$ and $\mathbf{t}_2$, and the vector-angle representations of the orientations are $(\mathbf{u}_1, \theta_1)$ and $(\mathbf{u}_2, \theta_2)$. We then compute the pose for each non keyframe $k \in \{k_1 + 1, k_2 - 1\}$ by linear interpolation:

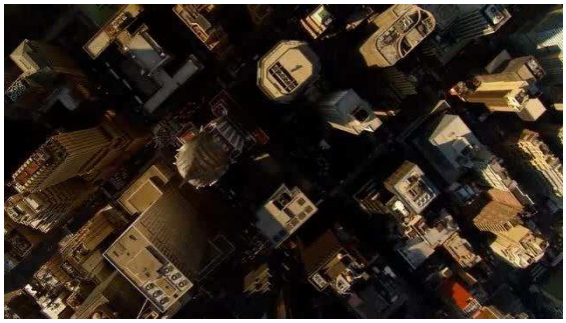
$$\left\{ \begin{array}{rcccl} \mathbf{t} & = & (1-\lambda)\,\mathbf{t}_1 & + & \lambda\,\mathbf{t}_2 \\ \mathbf{u} & = & (1-\lambda)\,\mathbf{u}_1 & + & \lambda\,\mathbf{u}_2 \\ \theta & = & (1-\lambda)\,\theta_1 & + & \lambda\,\theta_2 \end{array} \right. \text{ with } \lambda = \frac{k-k_1}{k_2-k_1} \tag{16}$$

Notice that for numerical safety reasons, the vector $\mathbf{u}$ is renormalized once interpolated to ensure its unitary property.

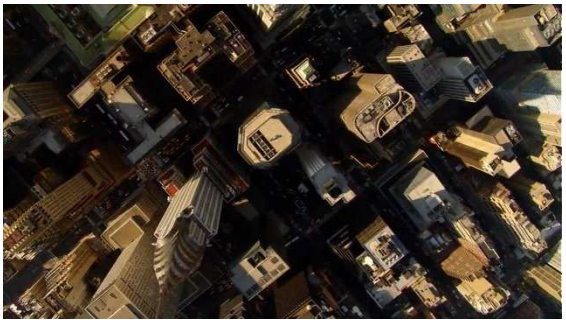
6.4.2 - Dense structure

From sparse points to depth maps Formally, a pixel in a depth map holds the Z coordinate of a point belonging to the associated line of view. This line passes through both the considered pixel and the camera center, and the Z value is expressed in the camera coordinate frame. Thus deriving a 3D point position associated to each pixel for an image is equivalent to have a depth map, up to a rotation and translation transformation to bring them in the camera frame.
During this project, we explored two different ways to reach this goal. Surprisingly, none of them take directly advantage on the already estimated 3D points, but rather recompute them at once for each pixel of each image. Here the existing structure has only been used as a strong and robust support to camera poses estimation. The key idea is rather the same than for 3D points computation: they are "triangulated" (see following section for the use of quotes here), which requires on one side two images and the associated camera poses, and on the other side matched features between images. Here all pixels have to be matched. Differently speaking, all we need now is the *motion flow* between the two images.
We now provide more details on how motion flow can be estimated, and how depth maps are derived in such context.

Rectification approach Traditionally, depth maps estimation in such context is performed *via* epipolar rectification. This rectification step constists in finding a transformation for an image pair constraining the epipolar lines to be parallel to image lines. Motion estimation is thus reduced to the 1D problem of disparities computation (see Part II).
We implemented three different methods to achieve such rectification: projective rectification [Har99], quasi-euclidean rectification [FI08] which minimizes images distortions, and polar rectification [PKG99],

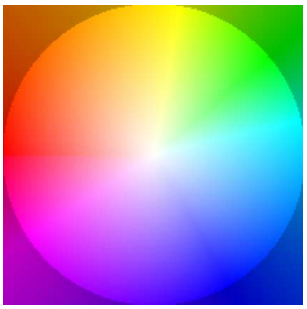


(a) I_{ref}



(b) I_{side}

(c) Motion flow



(d) Motion coding

Fig. 14. Optical flow estimation example for two images

which also minimizes transformed images distortions while allowing rectification in case epipoles are in the images (which is not the case for the two other methods).
However, such rectified images have to be coupled with a good stereo matching algorithm to compute the corresponding depth maps. Due to historical reasons, by the time these rectified methods were implemented, we did not have a good enough stereo matching algorithm, and thus turned to optical flow based methods.

Optical flow approach Another way to derive the motion flow of pixels is to directly use an optical flow algorithm. In [Gal02], such method is employed as a support for features tracking, and lately for depth maps retrieval. However, optical flow estimators were not efficient enough up to recently. For instance, motion at objects boundaries is often poorly estimated, leading to a blurred or smoothed motion, and as such to inaccurate depth contours in the final maps. However, this topic recently received greater interest, especially through the Middlebury evaluation framework [BSL$^+$07][2], and very efficient methods have been proposed.
The method we use is that of Werlberger *et al.* [WTP$^+$09]. The principle of the motion estimation with these works is presented in Section 9.1. An example of the estimated motion that can be computed is depicted in Figure 14. The color wheel represents the color-code of the motion: orientation of the motion vectors is attached to the color, while the amplitude is represented by the saturation of this color. As one can see, motion boundaries are correctly dealt with.

Assume such motion flow has been computed for an image pair (I_{ref}, I_{side}). Then for each pixel $\mathbf{x}$ in I_{ref}, we use its matched position $\mathbf{x} + \delta$ in I_{side} to triangulate its 3D coordinates $\mathbf{X}$, given the estimated poses $\mathcal{P}_{ref} = [\mathbf{R}_{ref} \mid \mathbf{t}_{ref}]$ and $\mathcal{P}_{side}$. Finally, $\mathbf{X}$ is transformed to be expressed in $\mathcal{P}_{ref}$ coordinate system:

$$\overline{\mathbf{X}} = \mathbf{R}_{ref}\,\mathbf{X} + \mathbf{t}_{ref} \tag{17}$$

[2] `http://vision.middlebury.edu/flow/`

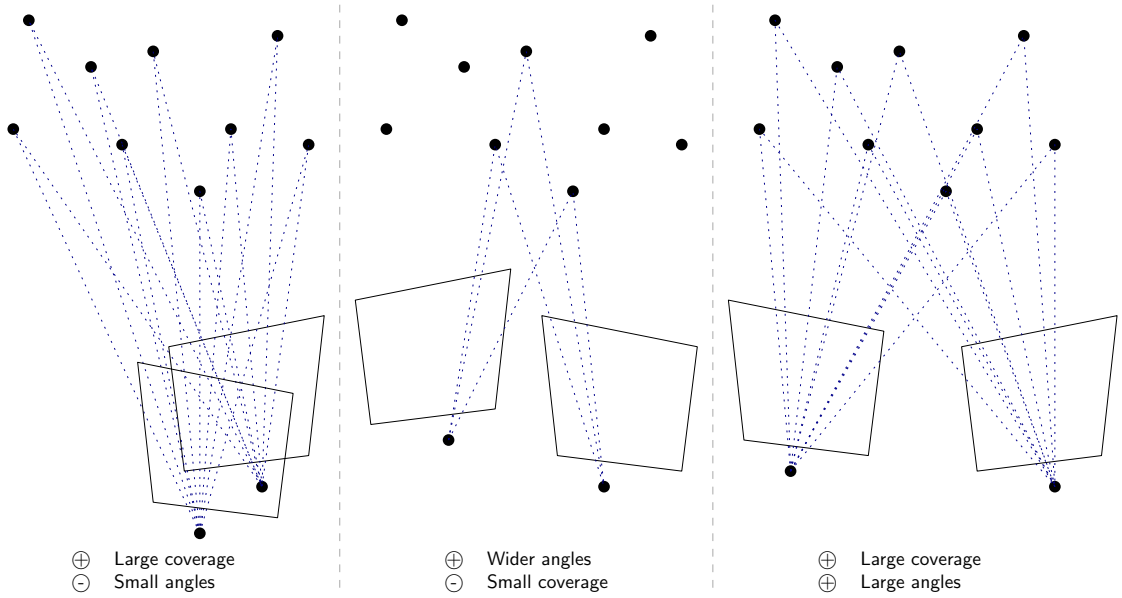


Fig. 15. Reconstruction quality criterion based on 3D space measurements.

The value stored in the depth map for pixel $\mathbf{x}$ is then simply the Z coordinate $\overline{\mathbf{X}}_Z$.

The real problem here is not how motion is computed or 3D points triangulated, but rather which images to choose to perform such task. Given I_{ref}, the problem translates to find I_{side}. This issue is similar to the keyframe selection algorithm, except for now the camera poses are known for each image. We propose now a criterion measuring the relevancy for images to be associated to I_{ref}, for a depth map computation purpose.
We search an image I_{side} in a local window W around I_{ref}, with $W = \{ref-N, ref+N\} - \{ref\}$ which maximizes a reconstruction quality objective function. We believe the baseline between cameras is not a relevant enough criterion to measure such future reconstruction quality, and we use instead a 3D space measure corresponding to the sum of the angles formed by already computed 3D points and the cameras positions, which are denoted here $\mathcal{C}_{ref}$ and $\mathcal{C}_i$. This is illustrated on Figure 15. On the left side, baseline is small, leading to poor reconstruction quality evaluation since 3D triangulation will be highly corrupted by noise. Notice considered angles are quite small. The right side depicts the contrary (and good) case. In the middle, we illustrate a camera configuration for which the baseline may seem sufficient, but which is not that good since images share small image common parts. For two given cameras the quality is thus measured as:

$$\begin{aligned} qual(\mathcal{C}_{ref}, \mathcal{C}_i) &= \sum_{j=1}^{M_i} \widehat{\mathcal{C}_{ref}\mathbf{X}_j\mathcal{C}_i} \\ \Leftrightarrow \quad qual(\mathcal{C}_{ref}, \mathcal{C}_i) &= \sum_{j=1}^{M_i} \frac{\overrightarrow{\mathbf{X}_j\mathcal{C}_{ref}} \cdot \overrightarrow{\mathbf{X}_j\mathcal{C}_i}}{\|\overrightarrow{\mathbf{X}_j\mathcal{C}_{ref}}\| \|\overrightarrow{\mathbf{X}_j\mathcal{C}_i}\|} \end{aligned} \tag{18}$$

with M_i the number of 3D points both seen from images I_{ref} and I_i. Notice that we do not compute the mean of such angles, but the sum of them for a given camera pair. This penalizes configuration such as the middle one on Figure 15. Finally, we have:

$$I_{side} = \underset{i \in W}{\operatorname{argmax}}(qual(\mathcal{C}_{ref}, \mathcal{C}_i)) \tag{19}$$

Depths scaling The last remaining problem concerns the scaling of depth maps. Indeed, such maps are generally stored and used as greyscale images. As such, the range of estimated depths should be scaled

to fit the $\{0, 255\}$ range of grey values. For now on, this scale is provided by the user, but it should be computed in an automatic way (for instance by analyzing the depths of already reconstructed points).

6.4.3 - Results

We present in this section some results of depth maps extraction from monocular videos. The figures we show represent a subset from first to last images of given videos, with associated depth maps. The four sequences presented are extracted from the movie *Home*. The source images are extracted from a 720p, 24Hz, and H.264 encoded video. They hold between 125 and 300 images. For none of these videos the internal camera parameters are known. They are thus initialized to approximate values.

In Figure 16, it is relatively hard to distinguish some zones within the images, on one hand because of the ice structure, and on the other hand because of the uniform sky. We can see that though ground and icy parts seem correctly modeled, the sky part is pushed towards some mean depth value. This is because 3D motion estimation here is very hard in such zones where points are at infinity and are completely uniform. Nevertheless, for near structures, the depths and discontinuities are well estimated, and consistent across images, though no temporal consistency has been enforced.

For the sequence in Figure 17, the viewed structure is somewhat binary: the ground (or at least the water surface) on one side, and the low altitude clouds on the other side. We wanted here to test our algorithm against data with no well determined contours (the clouds). Moreover, this video doesn't respect the simple pinhole camera model we use, without radial distortion. This distortion is quite important here with the short focal length used to acquire the video. We can notice that the zones corresponding to the ground are white within the depths maps. This is due to the fact that depth maps scaling parameterization has not been well done. However, this does not impact on the relief visualization of the video: the ground is modeled as a flat zone far away from the camera. We notice finally that dense cloud zones are correctly estimated, but their boundaries tend to be propagated towards their neighbors. This is partly due to the fact that zones between clouds, on the ground, are mainly uniform and 2D motion is propagated between them.

The video depicted on Figure 18 is easier to deal with. This appear clearly in the depth maps quality, especially around the discontinuities, all along the video. There is a small error however that can be noticed. In the last image, a small white dot appears on top of the ice part. It corresponds to the helicopter shadow, which violates the static scene constraint: it is a small moving object on the ground surface. As a consequence, it has a smaller 2D motion amplitude than the ground, and is estimated as much more far away from it, giving this large Z value.

The video presented in Figure 19 combines several aspects leading to a good depth maps estimation: smooth camera motion at constant speed, highly textured images and dense scene (compared to *Corail*). The depth contours are well delineated, and the relative depths correctly estimated. This is the typical video providing well conditioned input data to dense depth reconstruction.

6.5 - Conclusion and perspectives

We presented in this section a new *SfM* method built upon video inputs, to produce depth maps for each input image. It relies on recent state-of-the art algorithms, and provide relevant enough results to display the resulting 2D+Z videos in an auto-stereoscopic way (see Part III).

Several issues remain unsolved however, and shall be dealt with in upcoming works. First of all, It must be noticed that the optical flow-based method we present is less efficient than Galpin's one [Gal02] around the epipoles, when they are in the images. This can be explained by the fact that Galpin's method cumulates frame-to-frame 2D motion while our methods computes motion fields directly between an image pair. So at the epipole zone the motion is zero and no 3D information can be retrieved, while when cumulating motion, very small displacements around the epipoles can be detected.
Deeper studies shall also be conducted towards epipolar rectification and depth maps estimation relying on disparities instead of triangulation. Indeed, in this project rectification works have been performed

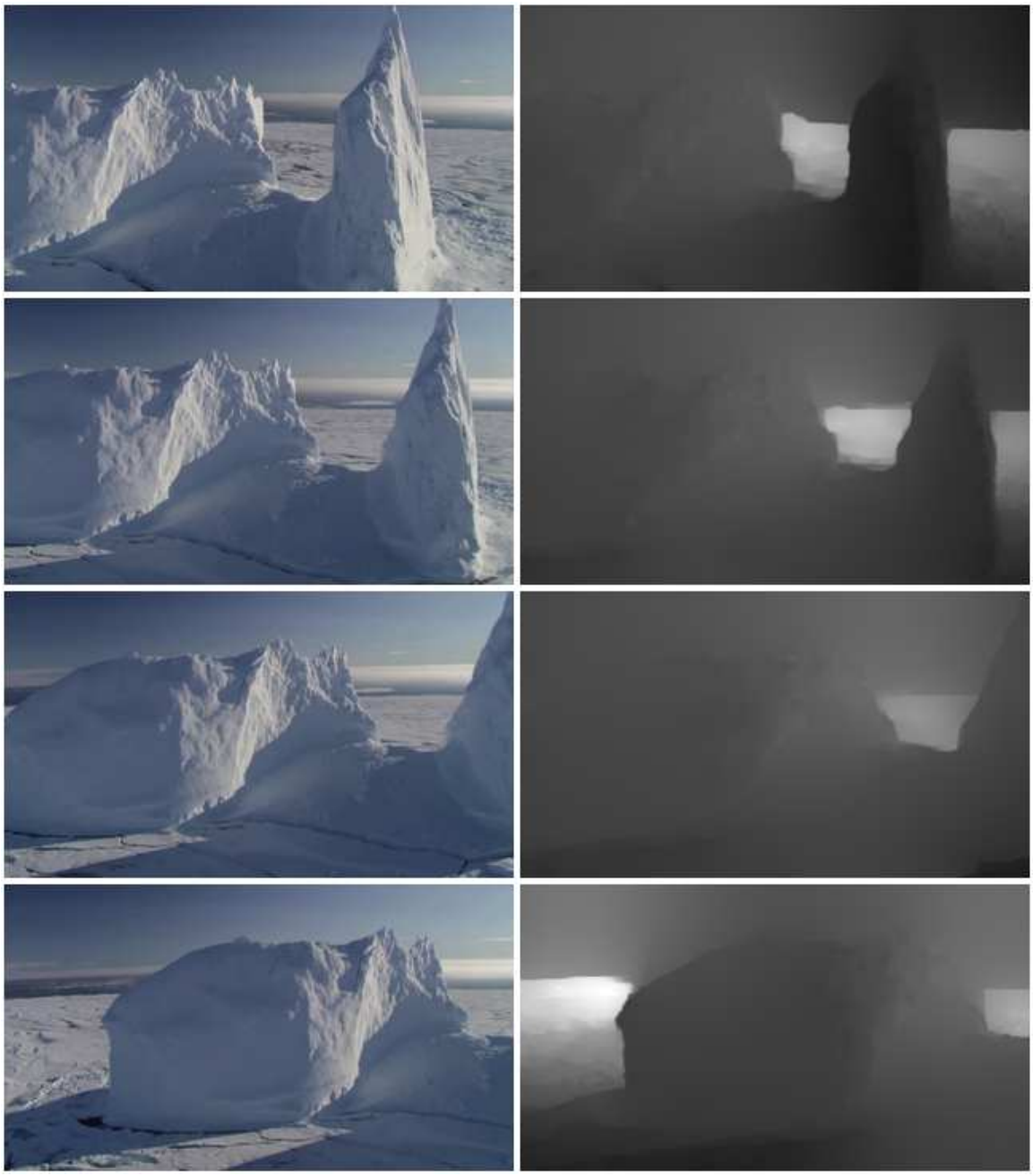


Fig. 16. Depth maps extraction for sequence *Home - Arctic*

before optical flow ones, and were intended to be associated with stereo matching algorithms. However, the results were too poor to be exploited due to the lack of precision in the rectification process. Such non quite exactly registered stereo pairs could be input to the optical flow algorithm to compute disparity maps, and thus depth maps. More precision could be attained in the epipoles zones using Pollefeys' polar rectification framework [PKG99].

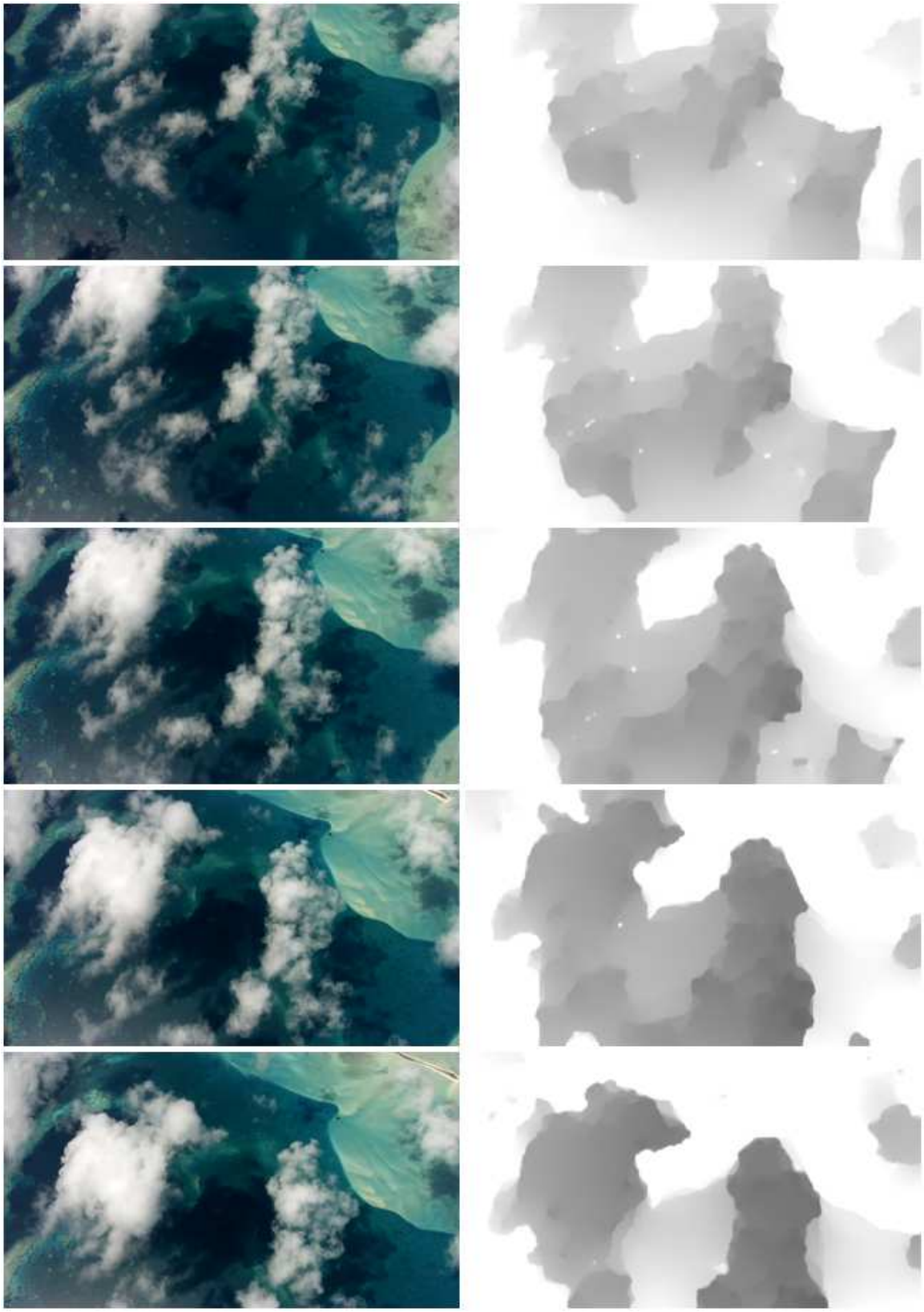


Fig. 17. Depth maps extraction for sequence *Home - Coral*

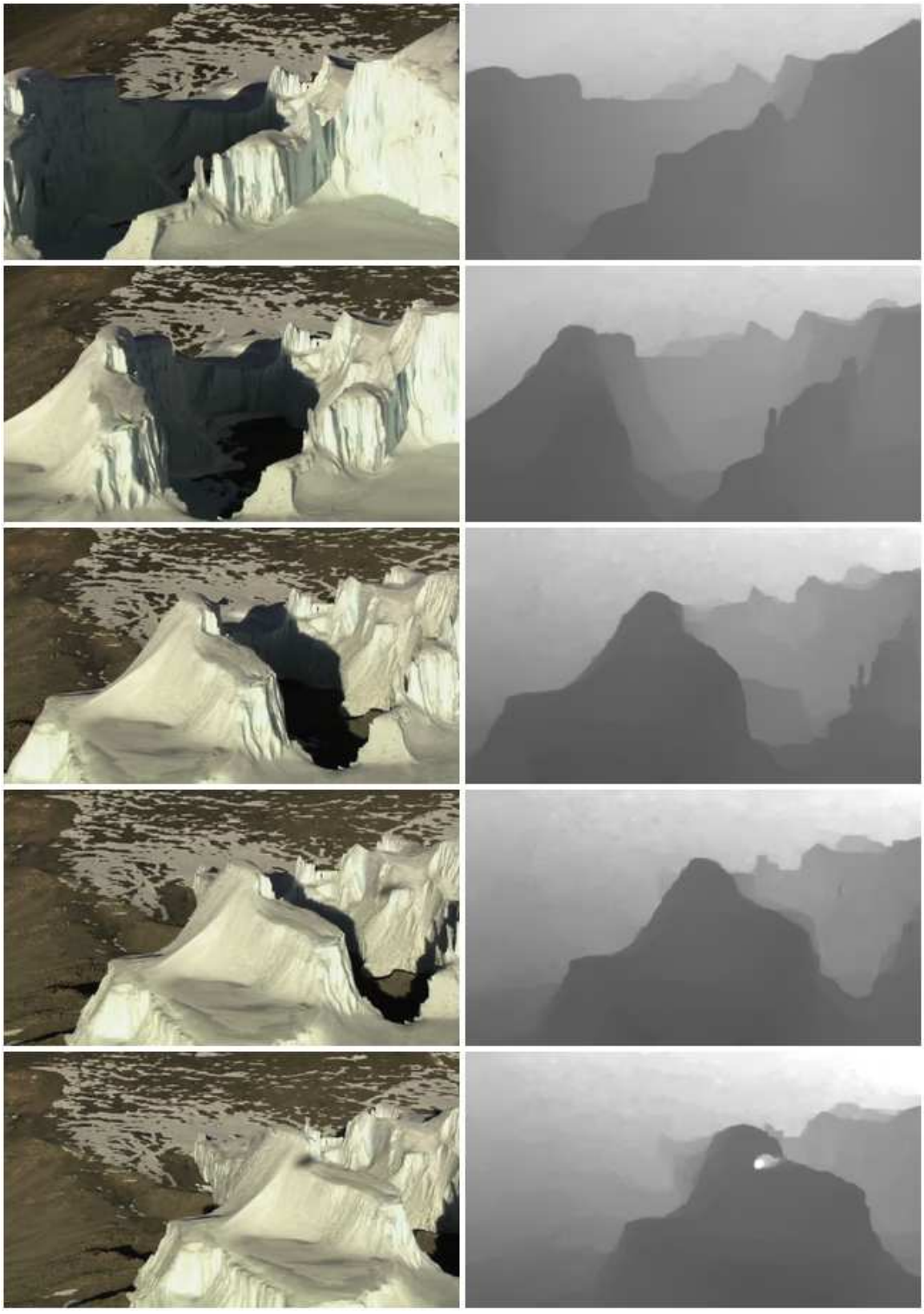


Fig. 18. Depth maps extraction for sequence *Home - Kilimanjaro*

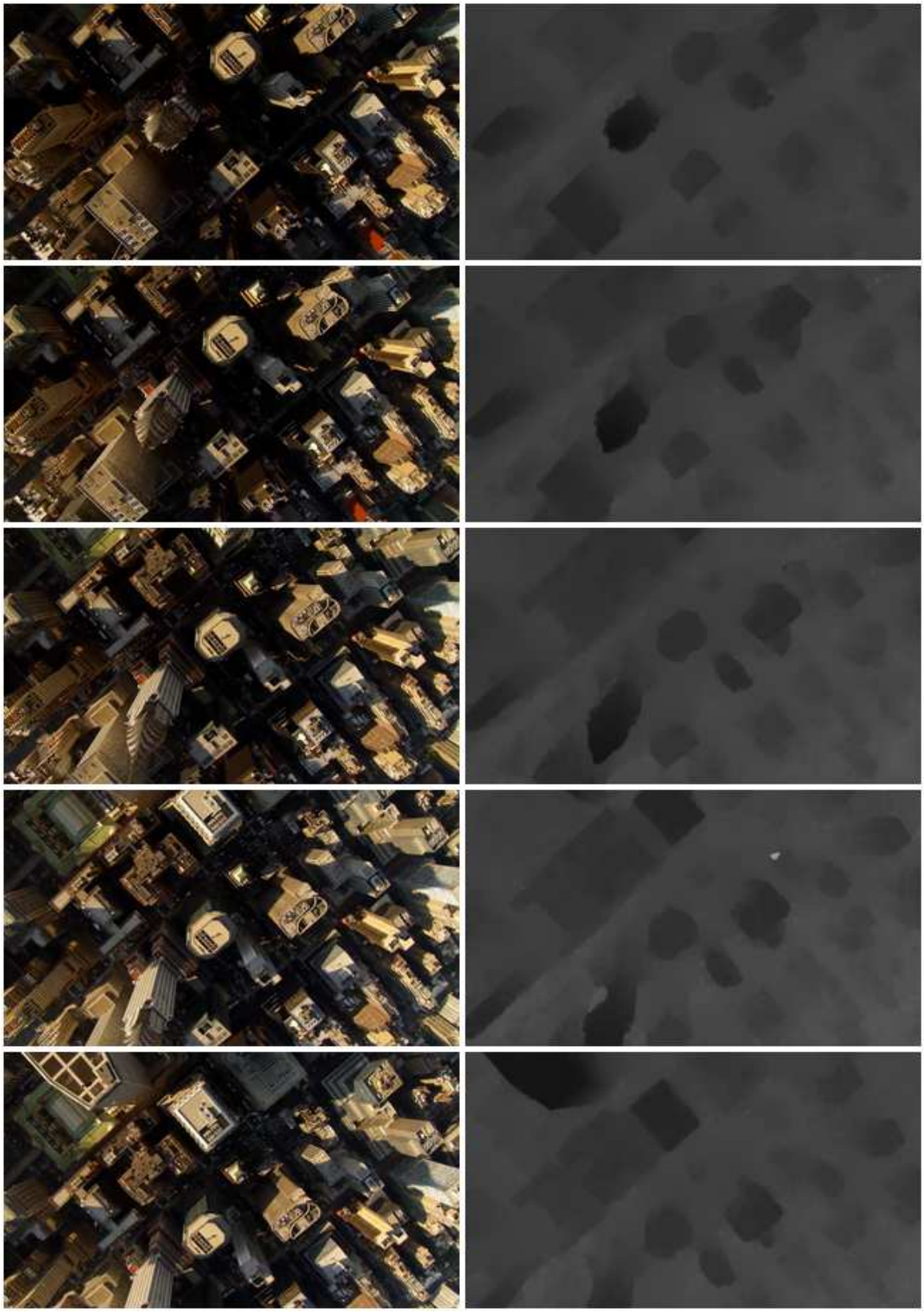


Fig. 19. Depth maps extraction for sequence *Home - New York*

PART II

Depth maps estimation for multi-view videos

7 - Introduction

We present in this part similar works than those described in the previous part, as we seek to compute depth maps for video content. However, these works differ by the use of multi-view input content. This is an essential problem in 3DTV. Indeed, depth maps represent the primary source of direct 3D information[3], for any video content.
In the future 3DTV broadcasting scheme, we know that many display types will be available, leading to many required input 3DTV content. Let's say, to take an extreme example, that two users want to see a broadcasted live TV show. The first one uses his mobile phone, requiring two different views of the scene. The second one needs 256 views to perform 3D display on his brand new and hypothetical holographic display, and, unluckily, none of them share any of the views they need. Does this mean that on stage there should be at least these 258 acquired points of view? Obviously not, and that's where depth maps become very handy. They are used to generate intermediate point of views.
On the other hand, if we compare with depth maps extraction from the previous part, we see that a single camera setup is not an option. It would require much more advanced frameworks to overcome the restraining hypothesis of moving camera, static scene, non deformable objects, etc. This is why multi-view videos are acquired in the very beginning.
We now describe with more details what multi-view contents are, the base principle of depth-disparity relationship, and give a few words on ongoing works towards normalization for depth maps extraction. In the following sections, we describe our methods to extract depths from such videos.

7.1 - Multi-view videos

We assume here the same acquisition context than the one proposed by the MPEG group. We consider that input videos are recorded by an aligned camera bank, with non converging cameras. Their image planes are thus aligned to the same virtual 3D plane (see Figure 20). As such, input views can not be used directly to perform auto-stereoscopic display.
We notice that such recording process is very difficult to set up, and thus input images are generally corrected to be perfectly aligned not only geometrically speaking, but also chromatically.

[3] As opposed to indirect 3D information, for instance a couple of stereoscopic images, which do not represent 3D information but can be used provide a 3D experience.

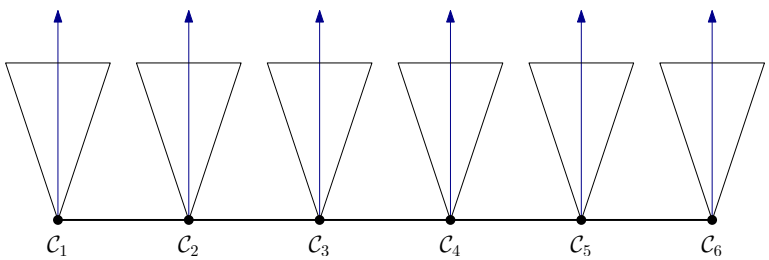


Fig. 20. Illustration of a camera bank. Cameras are aligned and point towards parallel directions.

7.2 - About depth and disparity

Given two images extracted from a multi-view video, as presented in the previous section, we generally seek to estimate the *disparities* between these images, before getting the related depth. These disparities correspond to the 2D motion flow amplitude between the two images. Let's take for instance the cameras $\mathcal{C}_1$ and $\mathcal{C}_2$ from Figure 20. $\mathcal{C}_1$ is associated to the left image while $\mathcal{C}_2$ is associated to the right one. The disparity map for $\mathcal{C}_1$ given the couple $(\mathcal{C}_1, \mathcal{C}_2)$ represents for each pixel in $\mathcal{C}_1$ the amplitude of the *horizontal* motion from this pixel to its match in $\mathcal{C}_2$.
This notion of horizontal motion is very important. It comes directly from the geometrical configuration of the cameras (image planes parallel to the baseline), which enforces matched points across images to be on the very same line. In projective geometry, it corresponds to an image pair for which the epipoles are at infinity, and epipolar lines aligned to image lines.
Disparity maps estimation being dealt with only along image lines, it is greatly simplified. This principle is applied since many years in the Stereo Matching community, where one seeks to find depth maps from stereo pair images.

It is important to notice that one may speak indifferently in the community of depth maps of disparity maps, though they do not encode the same information. This is due to the fact that the former are inversely proportional to the latter. For a given camera pair and their associated disparity maps, two additional information are required to retrieve the corresponding depth maps: the cameras focal length f (which should be the same for both cameras) and the baseline b between their optical centers. In case of non square pixels, and thus different values for vertical and horizontal focal lengths, the horizontal value is considered. For a pixel $\mathbf{x}$ with disparity d, the depth value can thus be computed in the following way:

$$z_{\mathbf{x}} = \frac{f * b}{d_{\mathbf{x}}} \tag{20}$$

This relationship can be explained using Figure 21, where data are depicted in blue while the unknown z coordinate is in red, with Thales' theorem.
It should be noticed that particular attention has to be paid to disparity values used for depths computation. A zero disparity corresponds to a point too far away from the cameras, so that its projected motion between the images is not measurable within the pixel grid. Such points are called *points at infinity* in this context, and are consequently associated to an infinite depth value. In a framework where depths are stored in a scaled image with a given min/max range, specific treatments should be applied to such points.

7.3 - Related normalization works

Ongoing works for future 3DTV complete framework normalization deal with several aspects:

- ↬ Acquisition of multi-view content
- ↬ 3D representation
- ↬ Intermediate views generation
- ↬ Coding / compression and transmission
- ↬ etc.

The main 3D representation in use is here the depth map. As such, extraction of this information has been addressed by the MPEG community under the form of a Depth Estimation Reference Software (DERS). This software transforms multi-view videos into multi-view plus depth videos (MVD). Evaluation of such generated depth maps is performed through virtual views synthesis using another reference software: VSRS, the View Synthesis Reference Software.

7.3.1 - Depth estimation

Depth maps estimation within the DERS is mainly inspired by the works belonging to the Stereo Matching community of the past few years. To simplify, disparities are estimated in three different steps [SS02]:

1. Local search of pixel matches along image lines

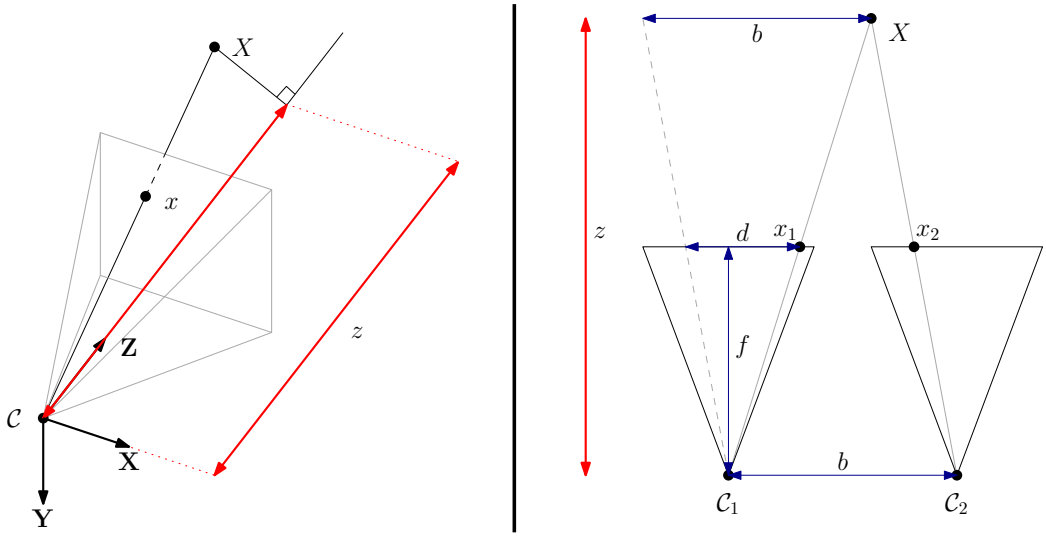


Fig. 21. Relationship between disparities and depths.

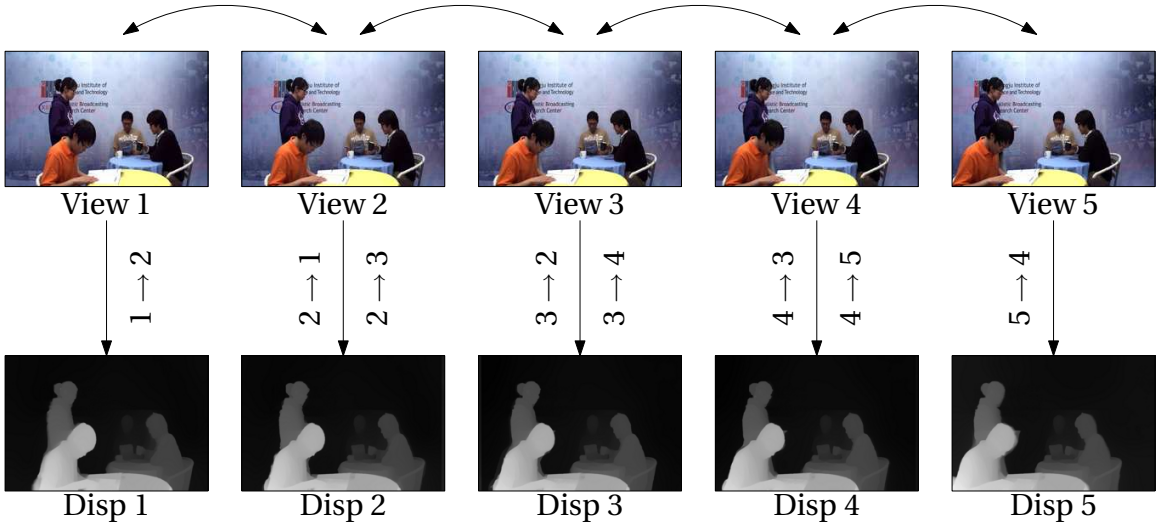


Fig. 22. Depth estimation framework for the DERS

2. Global optimization over the whole image
3. Post-processing

More details on the Stereo Matching principle are given in the following section. It has been adapted here to fit the requirements of the multi-view context. Instead of using only two views (left and right), the DERS considers three input view (left, central and right). The disparity map is computed for the central view using motion estimation from both the left and right views, to tackle efficiently with occluded zones for instance. Implicitly, this framework imposes that motions from left or right views be equivalent and thus that the three cameras are perfectly aligned, the central one being at an equal distance from the two others. This depth estimation framework for the DERS is illustrated on Figure 22 (here, for illustration purposes, we show depth maps for the sequence *Cafe* computed with our method, described in Section 9). For instance, the disparity map for view 3 is computed using pixels motion between view 3 and views 2 and 4.

The DERS cannot estimate disparity maps for all views at once. They are estimated for each views separately. Moreover, the DERS does not output depth values and these have to be computed from disparities

(a) *Newspaper* (b) *Book Arrival*

Fig. 23. Example of disparity maps extraction for two MPEG test sequences

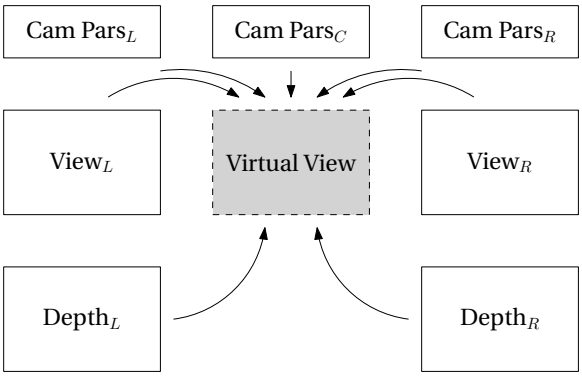


Fig. 24. Virtual view generation framework using the VSRS

if depths maps are required for another application. Finally, with such framework, disparities for extreme views can not be computed since a side view is missing.

Figure 23 illustrates depth maps extraction results for two test sequences: *Newspaper* and *Book Arrival*. Disparity maps are encoded in greyscale images: dark values indicate pixels far away from the cameras while bright values depict near objects. Depths maps for *Newspaper* are computed in an automatic way while manual disparity data have been integrated in the estimation for *Book Arrival*. Some disparities are badly estimated (*Newspaper*: pixels on the background above the left-most character are noted much nearer than they should do). Some inconsistencies between the maps can also be noticed (*Newspaper*: on the top-right part of the background ; *Book Arrival*: on the ground, to the bottom-right side).

This disparity estimation phase is crucial, since it impacts on all the remaining steps of the chain. To our point of view, the DERS did not behave well enough as is. We thus present in Section 8 more elaborated works on Stereo Matching which could be integrated to the DERS. In Section 9, another software similar to the spirit of DERS is presented. It is based on Werlberger's optical flow algorithm [WTP+09].

7.3.2 - View synthesis

Evaluation of disparity or depth maps can be performed using the following protocol. Two non neighboring views and their associated maps are considered. The VSRS is used to generate an intermediate view, corresponding to one of the acquired views. These two views — the generated one and the acquired one – are then compared using an objective metric. This metric can be for instance the PSNR, or the more recent PSPNR measure, which aims to consider perceptual differences between the images.

The VSRS uses as input data two videos and their associated disparity maps (which are converted internally to depth maps), corresponding to two different acquired views. It also needs the camera parameters of these two views (both intrinsic and extrinsic). This process is illustrated on Figure 24. A central view is generated, using the camera parameters which are desired for this virtual point of view and the other input data. No 3D information is generated with the VSRS, only 2D images.

8 - Stereo matching approach

Our Stereo Matching algorithm has been originally developed to compute depth maps for pairs of rectified images in the monocular context (see Part I). It could also be applied to the multi-view context in a similar way, except that here input images are already rectified. We review in this section the principles of out algorithm, together with evaluation results with stereo images from the Middlebury benchmark dataset[4]. Foreword note: These works have not been used *in fine*, either for the monocular case of the multi-view one. The optical flow-based framework is preferred in both cases. However, since the DERS is based on such methods, we though it would be interesting to describe related works we made on the subject.

8.1 - Framework

We quickly review here the essential concepts of the Stereo Matching framework. A much more complete review can be found in [SS02].
Stereo Matching can often be split into three successive steps :

1. Local matching
2. Global optimization
3. Post-processing

Input data consist in a set of two stereo-rectified images I_1 and I_2. The stereo-rectification constraint implies that matches across images are along the same image lines (see Section 7.2). The stereo matching process consists in finding, for each pixel (x, y) in, let's say, I_1, the corresponding pixel $(x \pm d, y)$ in I_2. The variable d represents the disparity for this pixel. It's sign depends on the left/right ordering of the images (see Figure 25). A local search space for d is defined, between a minimum and maximum disparity possible values: $d \in [d_{min}, d_{max}]$. To summarize, the search space used here is defined in $\mathbb{R}^3$ for each image. It is called the *Disparity Space Image* (DSI).
The local matching procedure consists in computing matching costs in the DSI for each pixel, in the neighborhood of this pixel. Generally, the disparity associated to a given pixel is the one with the lowest matching cost. The global optimization step's purpose is to regularize the DSI so that the disparities be more consistent over all the image. The post-processing can have several purposes: outliers filling, plane fitting to smooth disparity maps, etc.
We now explain with more details our Stereo Matching procedure.

8.2 - Local Matching

Our local matching procedure is based on the local self-adaptive method described in [KSK06]. Two different matching costs are combined to form the final matching cost in the DSI: the sum of absolute difference (SAD) and the sum of the gradient difference (GRAD). The considered neighborhood for each pixel is

[4] `http://vision.middlebury.edu/stereo`

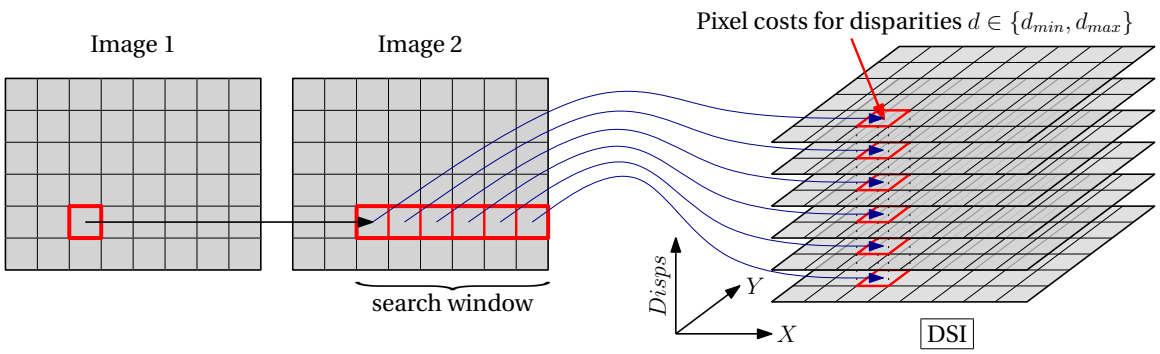


Fig. 25. Disparity search space principle

a 3×3 block centered on the aforementioned pixel.

$$\text{DSI}_{\text{SAD}}(x,y,d) = \sum_{i=-1}^{1}\sum_{j=-1}^{1} |I_1(x+j,y+i) - I_2(x+j\pm d, y+i)| \tag{21}$$

$$\text{DSI}_{\text{GRAD}}(x,y,d) = \sum_{i=-1}^{1}\sum_{j=-1}^{1} \nabla I_1(x+j,y+i) - \nabla I_2(x+j\pm d, y+i) \tag{22}$$

The final cost stored in the DSI for a given pixel is computed by optimizing a weighting coefficient α with regards to a cross validation criterion. We assume here α is given. The way to compute it automatically is presented below. Given α, we have:

$$\text{DSI}_\alpha = (1-\alpha)\cdot \text{DSI}_{\text{SAD}} + \alpha \cdot \text{DSI}_{\text{GRAD}}, \quad \text{with } \alpha \in [0,1] \tag{23}$$

A disparity map $\mathcal{D}$ can then be computed by assigning to each pixel the disparity corresponding to the minimal cost in the DSI. This is a *Winner Take All* (WTA) strategy:

$$\forall (x,y),\ \mathcal{D}(x,y) = \underset{d}{\text{argmin}}\ \text{DSI}_\alpha(x,y,d) \tag{24}$$

Since we have an image pair, this disparity map can be computed for both images. We can then measure the amount of consistent disparity pixels across the images. This is the *cross check* test.

$$\begin{aligned} N &= \sum_{(x,y)} \Psi(\mathcal{D}_1(x,y), \mathcal{D}_2(x,y)), \\ \text{with } \Psi(\mathcal{D}_1(x,y), \mathcal{D}_2(x,y)) &= \begin{cases} 1 & \text{if } |\mathcal{D}_1(x,y) - \mathcal{D}_2(x \pm \mathcal{D}_1(x,y), y)| < \lambda \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{25}$$

This relationship denotes the fact the displacement from I_1 to I_2 should be the same than from I_2 to I_1. Differences should only occur on objects and image boundaries, where pixels are viewed in one image but not the other. The parameter λ is a tolerance threshold on the motion difference, and is generally set to 1 pixel.
Coming back to our α value, the final DSI is computed by optimizing α such that it maximizes N:

$$\text{DSI} = \underset{\alpha}{\text{argmax}}\ N \tag{26}$$

The method used to optimize α is not described in the original paper. However, by computing the value of N for a high number of α samples, and several image pairs, we found out that the N function's derivative was almost monotonic and decreasing, meaning that N has one maximum, for all our data sets. This is illustrated on Figure 26 for four stereo pairs. As a consequence, to solve for α in fast way, we sample ten times the α range, take the maximum value α_1, and take another ten shorter samples around α_1 to refine the maximum value. It could be of course computed in a more elegant way through non linear optimization (with Levenberg-Marquardt or Newton-Raphson) without special care for initialization, since local minima are assumed to be very close to the optimal solution. It has to be tested to see whether it converges in less than 20 iterations, in which case it will replace efficiently our procedure.

8.3 - Global optimization

So as to limit noise inherent to local matching and enhance disparity results, the DSI's costs are optimized for the whole image using hierarchical belief propagation [SZS03]. For comparison purposes, the DERS uses a graph-cuts based global optimization approach [KZ01], which is known as slower and less efficient than hierarchical belief propagation in this context [SZS03, YWY+06].
At this point, information stored in the DSI are no longer used. The disparity maps are deduced using the WTA approach, and a post-processing step can be applied to enhance the final results.

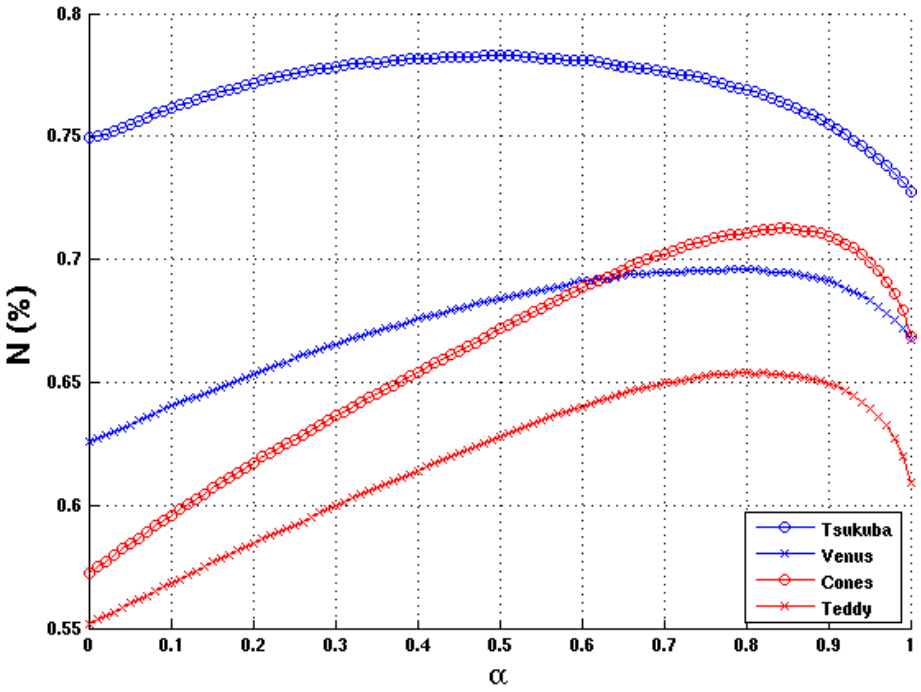


Fig. 26. Cross check quality *vs.* α

8.4 - Post-processing

One very important thing to keep in mind while enhancing disparity maps is that not all pixels are valid. As already stated, some of them can not be defined since there are parts of images that are only visible in one of them. These pixels are noted as outliers. One simple way to detect outliers is to apply the cross check test. Outlier pixels are those which violate the Ψ criterion (Equation 25).

With the assumption that the viewed scene is piece-wise planar, and that depth discontinuities occur at color boundaries in input images, one can apply segment-based plane fitting to smooth disparities in uniform areas while preserving depth contours. Input images are over-segmented with a mean-shift procedure [CM02], and a robust plane fitting algorithm is applied independently on each segment, without taking into account detected outliers. This procedure is based on voting, and is described in [WZ08].

A last outliers detection step can be performed, provided holes are correctly filled using neighboring inlier pixels.

8.5 - Results

We present here some disparity maps results from stereoscopic image pairs extracted from the Middlebury database. Each figure, from 27 to 36 is composed of four sub-images. The two top images depict the stereo pair used as input. The right bottom image represents the ground truth disparity provided in the database, corresponding to the upper left image. The bottom left image represents the disparity map computed with our method.
For each stereoscopic pair, two disparity maps are outputted. Computation times vary form thirty seconds to two minutes approximately. The validity percentage for the displayed disparity map is also reported (figures are given in the illustrations captions). This validity measure represents the portion of pixels for which the difference between the estimated disparity and the ground truth disparity is lower than one pixel. Since our algorithm is fully symmetric, results are very similar for the non shown right images. This percentage varies from 60% to 98.5%, the majority of the results being close to 85% valid

pixels. The low score for the *Plastic* sequence (61.23%) seems to be due to a shift in the disparity values. The disparity map itself is globally consistent with the ground truth (see Figure 35).
Some pixels are shown in black in the disparity maps. In the ground truth maps, they correspond to unknown pixels, which are only visible in a single image. In our results, they correspond to segments for which a disparity plane equation could not be computed, due to a large majority of outlier pixels in the given segment. In such case, inpainting methods should be applied.

8.6 - Discussion

The Stereo Matching method we presented in this section provides satisfying results with the state of the art database, but suffers from large computation times. However, input data are here very well calibrated, and the results may not be this good for instance with rectified stereo pairs extracted from a monocular sequence. Moreover, the outlier removal phase is not currently compensated by an efficient disparity prediction for missing pixels.
For these reasons, we explored another approach, based on new optical flow estimation algorithms. This method is presented in the following section.

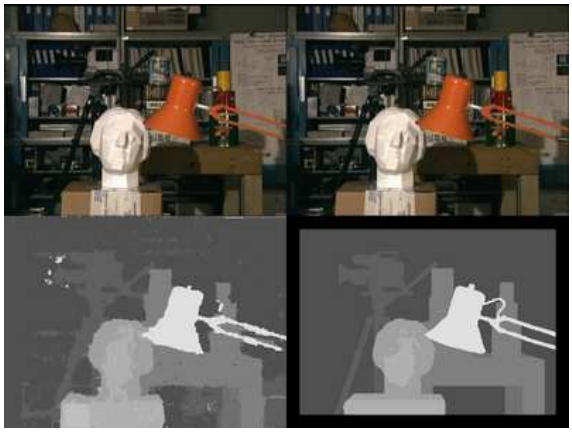


Fig. 27. Disparities for Tsukuba - 32.2s. - 96.67%

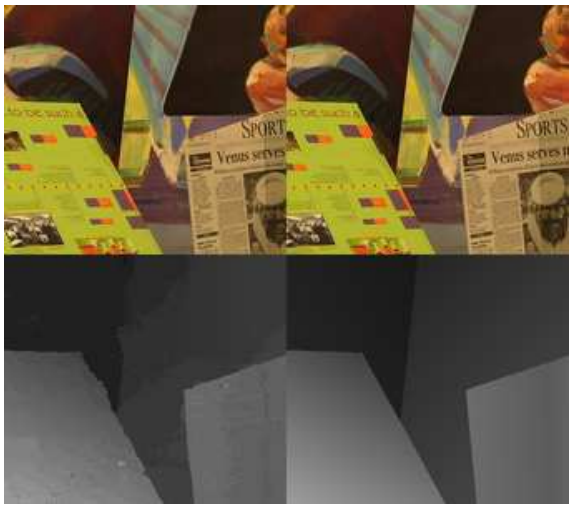


Fig. 28. Disparities for Venus - 63.1s. - 97.57%

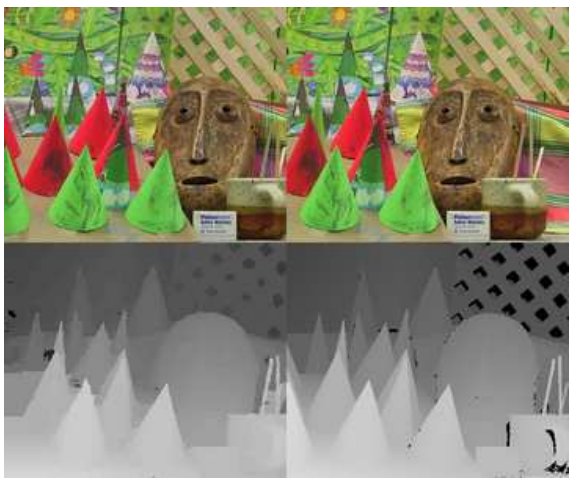


Fig. 29. Disparities for Cones - 104.2s. - 89.09%

Fig. 30. Disparities for Teddy - 94.0s. - 85.39%

Fig. 31. Disparities for Art - 115.0s. - 82.45%

Fig. 32. Disparities for Moebius - 116.0s. - 81.84%

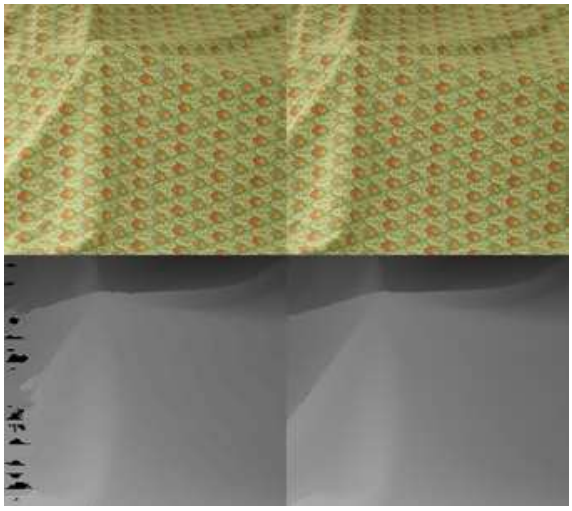


Fig. 33. Disparities for Cloth 1 - 81.1s. - 95.38%

Fig. 34. Disparities for Cloth 3 - 85.6s. - 95.12%

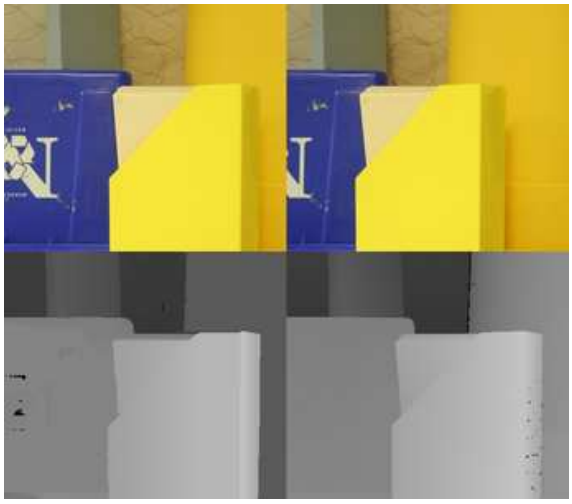


Fig. 35. Disparities for Plastic - 108.6s. - 61.23%

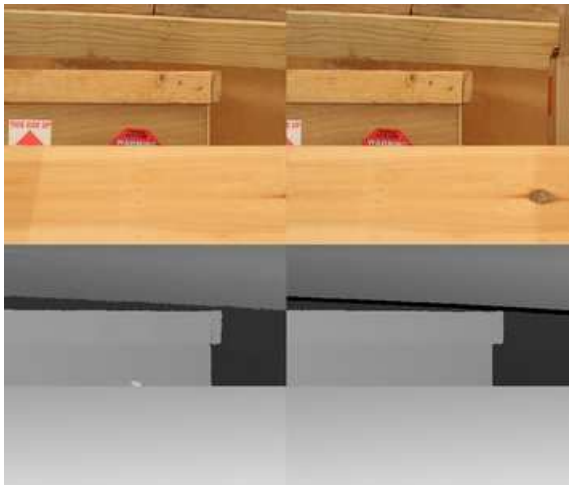


Fig. 36. Disparities for Wood 2 - 119.8s. - 98.84%

9 - Optical flow approach

9.1 - Principle and Werlberger's method

Based on the observation that a disparity field estimation is nothing else but a dense motion estimation between two images, we explored a new path towards recent optical flow estimation methods. These optical flow algorithms are a part of the 2D motion estimation algorithms. Their particularity comes from the fact that they seek to find the projected relative motion of scene points with regards to the cameras, which should be the closest possible to the "true" projected motion. With such definition, they can be opposed to motion estimation methods used in the video coding community, which aim at finding motion vectors that minimize an motion-compensated image difference in a local search window. For instance, in large textureless regions, if all motion vectors are equivalent in terms of image difference, they will favor null vectors since they are easier to encode, which will not represent the true 2D motion.

We now derive the Optical Flow Constraint (OFC) in a more formal way, going to Werlberger's formulation. The essential principles are explained. Much more details can be found in [Tro09]. The OFC comes from the brightness consistency between two images I_1 and I_2. The brightness of a pixel $\mathbf{x}$ in I_1 should be equal to the brightness of the matching pixel displaced by a motion vector $\mathbf{u}(\mathbf{x})$ in I_2:

$$I_1(\mathbf{x}) = I_2(\mathbf{x} + \mathbf{u}(\mathbf{x})) \tag{27}$$

By linearizing this brightness consistency constraint with a Taylor expansion, and dropping the negligible second- and higher-order terms, one gets the OFC:

$$\mathbf{u}(\mathbf{x})^\top \nabla I_2(\mathbf{x}) + I_2(\mathbf{x}) - I_1(\mathbf{x}) = 0 \tag{28}$$

Horn & Schunk showed that solving for $\mathbf{u}$ can be performed in an energy minimization framework [HS81]:

$$\bar{\mathbf{u}} = \underset{\mathbf{u}}{\operatorname{argmin}}\, E(\mathbf{u}) = \underset{\mathbf{u}}{\operatorname{argmin}}\, \left(E_{\text{data}}(I_1, I_2) + E_{\text{prior}}(\mathbf{u})\right) \tag{29}$$

Starting from this model, one can setup the energy formalization as a disparity preserving and spatially continuous formulation of the optical flow problem, based on a L^1 data term and an isotropic Total-Variation regularization term [PBB+06, ZPB07]:

$$E_{\text{data}}(I_1, I_2) = \lambda \int_\Omega |I_2(\mathbf{x} + \mathbf{u}(\mathbf{x})) - I_1(\mathbf{x})|\, dx\, dy \tag{30}$$

$$E_{\text{prior}}(\mathbf{u}) = \int_\Omega |\nabla \mathbf{u}_x| + |\nabla \mathbf{u}_y|\, dx\, dy \tag{31}$$

Here Ω represents the image domain, and $\nabla \mathbf{u}$ is the spatial gradient of the motion field.
By linearizing the data term, one gets a convex optimization problem:

$$E_{\text{data}}(I_1, I_2) = \lambda \int_\Omega |\rho(\mathbf{u}(\mathbf{x}))|\, dx\, dy, \tag{32}$$

with $\rho(\mathbf{u}(\mathbf{x}))$ being the Optical Flow Constraint from Equation 28. Such convex formulation ensures the minimizer to find the global minimum of the energy functional. Finally, to make their algorithm even more robust, [WTP+09] introduce an anisotropic (*i.e.* image-driven) regularization based on the robust Huber norm [Hub81].

Another extension of their approach consists in integration in the flow estimation not only the current image and the following, but also the previous image. The goal, in the original publication, is to cope with single degraded images within a video, for instance with historical video material.

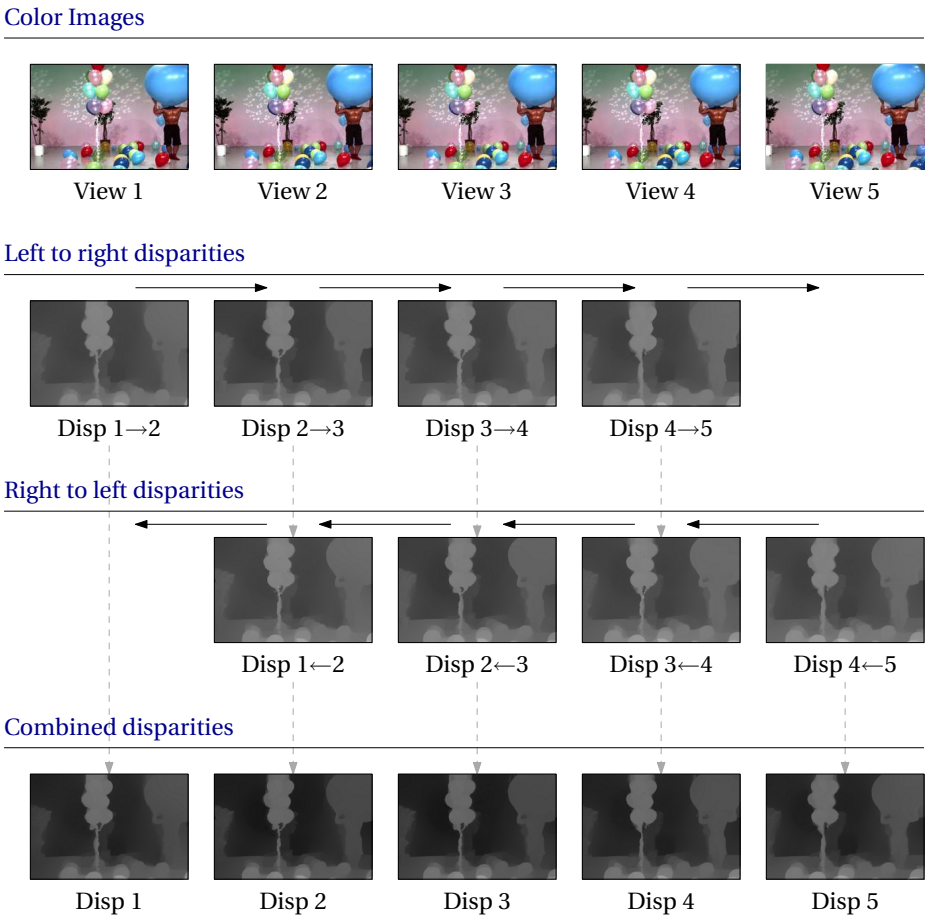


Fig. 37. Global framework of disparity estimation with *mv2mvd*

9.2 - Using optical flow in a MVD context

In this section, we present a software based on optical flow estimation explained in the previous section, designed to directly convert multi-view videos to multi-view videos plus depth. It is called *mv2mvd*.

Contrary to the DERS, it computes the disparity and / or depth maps for all views in one single pass. It's core uses the CUDA-based library[5] developed in parallel with [WTP+09]. The core framework of *mv2mvd* is depicted on Figure 37. One one hand, disparities are computed from left to right views (Figure 37, second row). On the other hand, they are estimated from right to left (Figure 37, third row). The interest of such two-side computation is to be able to locate occlusion zones where the motion field would be incorrect (a pixel in one view would not be visible in the other view). In fact, cross check is performed (see Section 8.2) to detect outlier pixels in each computed disparity map, which are finally combined (Figure 37, fourth row) by taking the minimal value of each disparity pair, to avoid the foreground fattening effect [SS02] exhibited by window-based algorithms.

9.3 - Results

We present in Figure 38 disparity maps extracted with out method for the sequences *Newspaper*, *Book Arrival* and *Lovebird 1*, together with original images and disparity maps computed with the DERS and provided to the MPEG community. Each line in the figure is related to one of the original views. The original images are in the first column, disparity maps computed with *mv2mvd* are in the second one,

[5] See `http://www.gpu4vision.org`

while DERS-based maps are in the third column. For the sequence *Lovebird 1*, notice that the gamma of the images has been modified for display purposes in this document (with the same amount for both maps types).

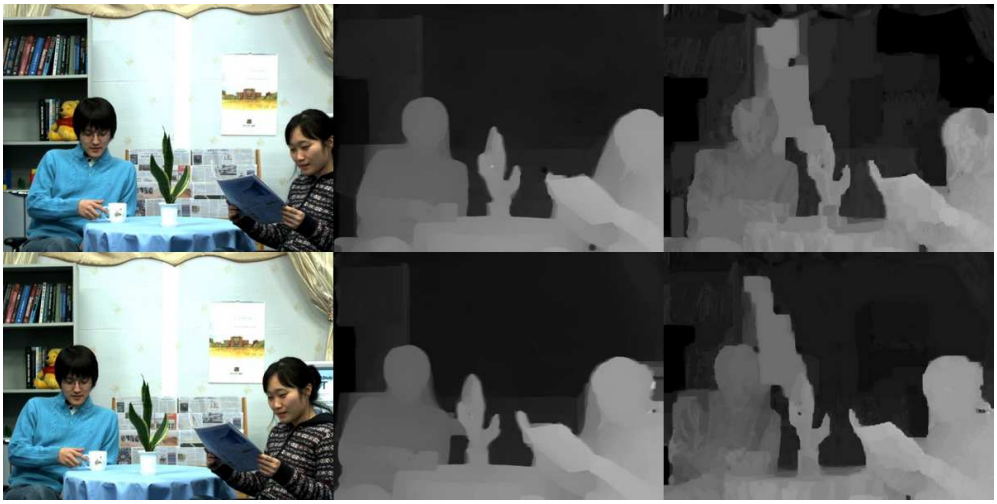
We remind that contrary to the DERS, results are computed for all desired views in a single pass. We can notice that globally, estimated disparities seem perceptually more relevant with regards to the scene with our approach. For instance, for the sequence *Newspaper*, the background of the scene is correct. With the DERS, numerous zones with different depths appear, while the depth of the background is physically the same with regards to acquisition cameras. As for the sequence *Book Arrival*, we can notice a greater spatial stability in the estimated depths, which appear more noisy in the DERS case. This comes for the latter from the application of plane fitting on mean-shift segments, which break the local spatial depth consistency applied to each segment. Finally, for the *Lovebird 1* sequence, one can see that out method is better suited to get depths variations and contours (see the building in the background, which appears more quantized in the DERS version). This sequence is quite hard to tackle since it holds a very wide range of disparities, from the foreground characters to the background buildings.

On Figure 39, we show visual differences between our optical flow-based disparities, and disparities deduced from a depth camera (z-cam) acquisition of the scene. These results are presented for the central of the five input views of the *Cafe* sequence. How such z-cam acquisition has been performed is described in [LKJH10]. Keeping in mind that these z-cam-based disparities are not raw and have been interpolated to fit the full video resolution, it is worth notice that our method competes very well with — and sometimes outperforms — the depth sensors. For instance, depth contours seem sharper with our method (all sub-images). We are even able to retrieve small depth details with much less noise (bottom right sub-image, for the chair part). However, for uniform zones with depths gradients, disparities are better estimated with the z-cam (see the furthest table for instance), where our method discretizes too heavily the resulting signal, while (again) better retrieving depth contours.

On Figures 40, 41 and 42, we present evaluation results of our disparity maps in terms of virtual views synthesis quality. The evaluation protocol used is the one used by the MPEG community.
Disparity maps are computed for views $N-1$ and $N+1$. These maps are used as input to the VSRS in order to synthesize the view N. This virtual view is then compared to the original view N in terms of PSNR, spatial PSPNR and temporal PSPNR, with the *Pspnr* tool provided to the MPEG community. We present for each sequence and each of these three measures three different plots (quality measure against video frame number). The red curve is associated to disparity maps generated by the DERS. The blue and black curves are respectively associated to our method without (MV2MVD F2) or with (MV2MVD F3) the integration of the symmetry constraint (see Section 9.1). The dashed horizontal lines in the figures correspond to the mean values over the whole considered sequence.
We notice that from now on, it is difficult to bring a clear conclusion to this evaluation procedure. Indeed, the quality of synthesized views seem to depend mainly on the input data. An ordering of the different methods per sequence is provided in Table 2. It appears however that our method seem to better behave in most of the cases than the DERS. We must also notice that compared to the DERS, there is absolutely no temporal consistency enforced in our algorithm, since it seems to provide stable enough results from one instant to the other, which is not the case of the reference software.

Sequence	**1st**	**2nd**	**3rd**
Newspaper	MV2MVD F3	DERS	MV2MVD F2
Book Arrival	MV2MVD F2	MV2MVD F3	DERS
Lovebird 1	MV2MVD F3	MV2MVD F2	DERS

Table 2. Ordering of methods by synthesized views quality



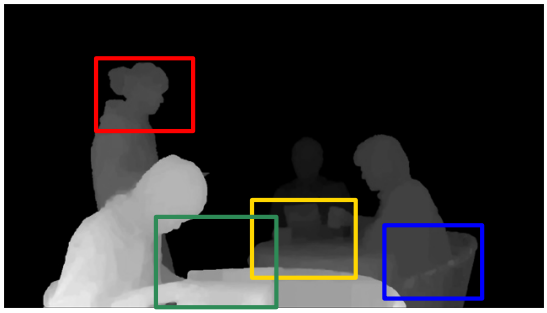
(a) Sequence *Newspaper*

(b) Sequence *Book Arrival*

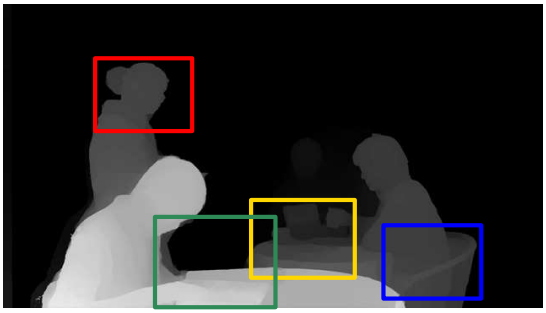
(c) Sequence *Lovebird 1*

Fig. 38. Comparison of extracted disparity maps between DERS and our method

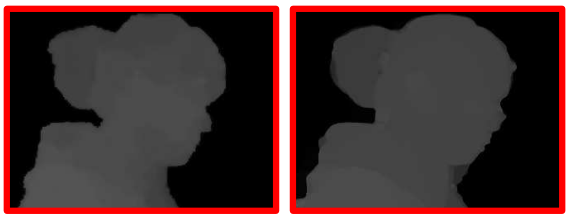
Color Image



Z-Camera (zc)

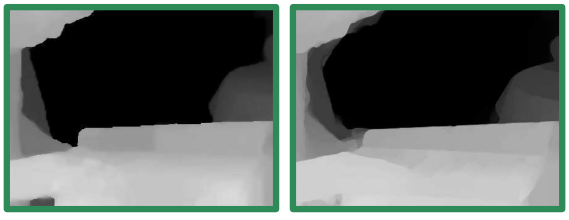


Optical Flow (of)

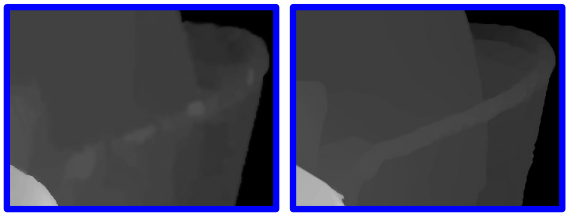


zc of

zc of

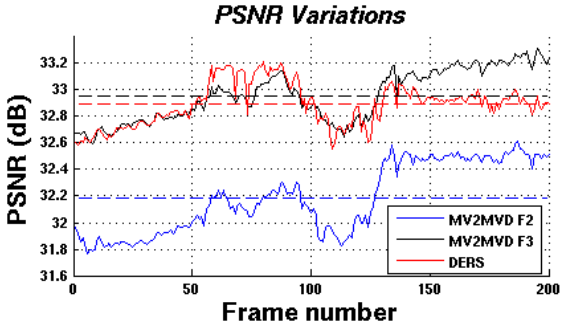


zc of

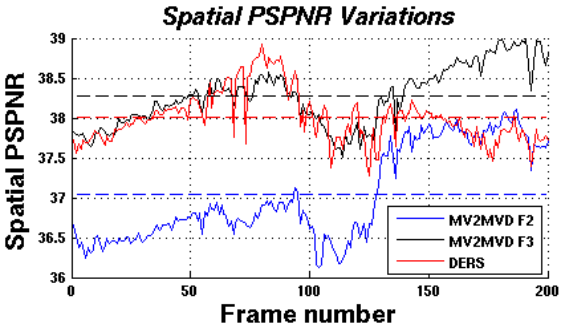


zc of

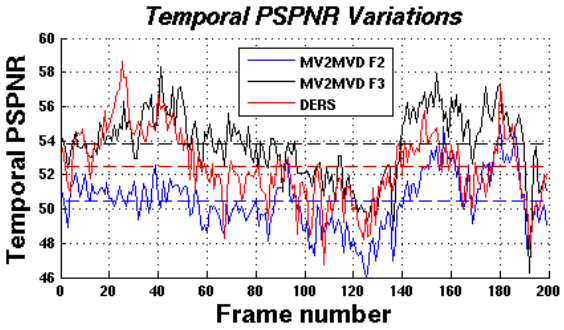
Fig. 39. Comparison between Z-Camera- and Optical Flow-based disparities



(a) PSNR

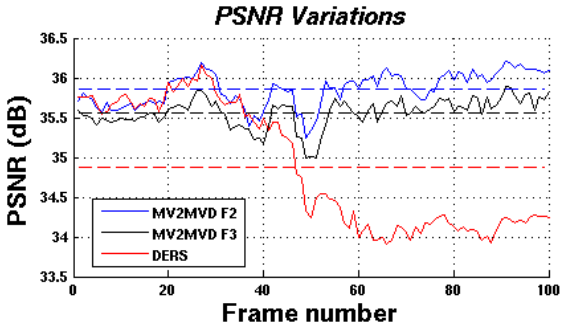


(b) Spatial PSPNR

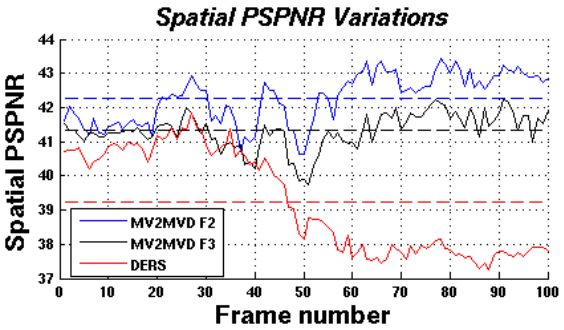


(c) Temporal PSPNR

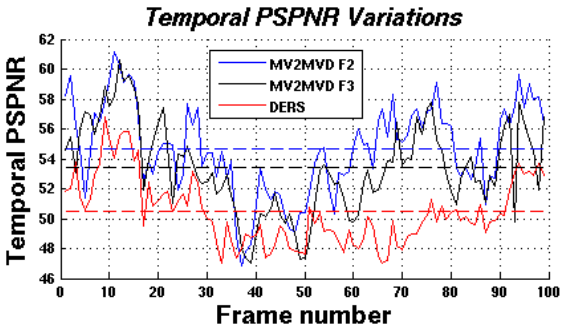
Fig. 40. Virtual view evaluation for *Newspaper*



(a) PSNR

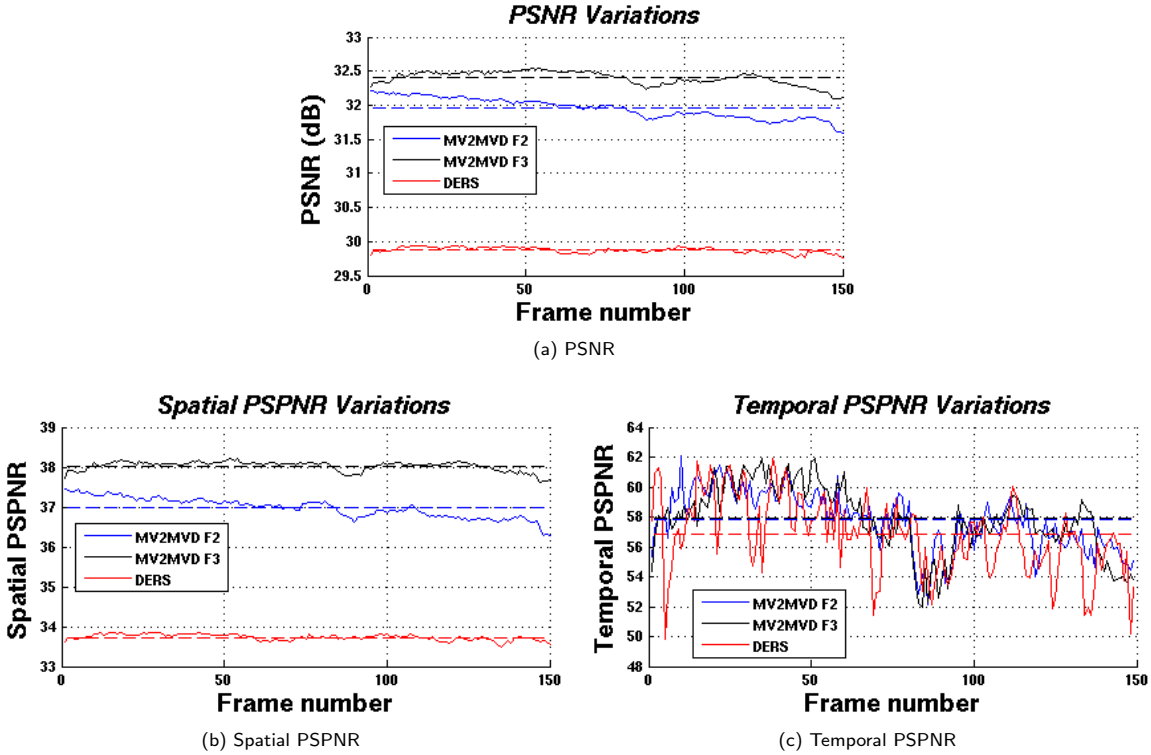


(b) Spatial PSPNR



(c) Temporal PSPNR

Fig. 41. Virtual view evaluation for *Book Arrival*



(a) PSNR

(b) Spatial PSPNR

(c) Temporal PSPNR

Fig. 42. Virtual view evaluation for *Lovebird 1*

10 - Conclusion and perspectives

We presented in this part a set of tools allowing the computation of depth maps for multi-view videos, on one side by using methods described in the stereo matching literature, and on the other side by exploiting recent advances in optical flow estimation.
The generated maps have been evaluated in terms of synthesized views quality using the VSRS reference software. It appears that our method gives promising results compared to maps computed by the associated reference software for depths extraction (the DERS). However, these results are subject to many interpretations and both methods are hardly comparable for the following reasons:

- No temporal consistency enforcement is integrated in out method, contrary to the DERS. This is due to the fact that such temporal consistency in the DERS can be interpreted as based on the assumption that the cameras are fixed, which we believe is way too restrictive.
- We do not propose to integrate depth masks during the estimation to provide manual data enhancing the final results, contrary to the DERS.
- At writing time, we were not able to constrain the optical flow estimation to be performed along image lines, as it should be with correctly rectified input stereo or multi-view images. We only select the horizontal component of the computed motion flow.
- Our method is only based on the luminance component of the input images, not on the RGB space, contrary, again, to the DERS.

Despite all these limitations with regards to the DERS, our method is able to compute depth maps totally relevant in terms of virtual views synthesis. Moreover, being implemented on the GPU, it is far faster that the DERS. The computational time can be reduced from 15 times to 150 times depending on the method used to compute the disparities with the reference software. And lastly, on an ease of use vision, our method computes maps for all views when a DERS execution is only valid for a single view, and has to be run independently for all of them.

PART III

Depth maps uses

11 - Depth-based auto-stereoscopic display

In the previous parts of this document, we saw that depth maps could be of several uses. They can be exploited to build a more advanced 3D representation of the scene, to add information used to enhance multi-view video coding, etc. In this section, we show that depth maps can also be used to add a relief experience to the video viewing, with auto-stereoscopic displays. We thus explain briefly how such displays work, how one can implement a rendering engine dedicated to such display, and finally detail how depth maps can be integrated to this purpose.
An extensive and comprehensive analysis of auto-stereoscopic displays can also be found in [MBBB07], from which several figures in this section come.

11.1 - The auto-stereoscopic principle

The stereoscopy principle itself is very simple: one has the sensation of viewing the scene in three dimensions if both eyes receive different signals corresponding to two different — and well calibrated — points of view of this scene. Thus, there must exist a device between the display and the user eyes to split the video signal. Traditionally, one uses glasses.

Auto-stereoscopic screens The auto-stereoscopic principle relies on the fact that the user doesn't need glasses anymore. The "signal separation" device is deported on the display screen itself. That's why one speaks of *auto-stereoscopic displays*: they are classical screens covered with a fine separation layer. This principle implies that images displayed on the screen must be of a specific nature: they will be split into several sub-images sent to each users' eyes. So the different images that will be sent to the user(s) are interlaced into one single image that is finally separated and sent to different eyes.

Multi-users displays One thing to notice about auto-stereoscopic displays is that they are not necessarily meant to be viewed by a single user. They are intended to display scenes in 3D with enough comfort for several users.
Let's assume from now on that such display only sends two different views to the users. The signal separation creates in the space in front of the screen what we call *viewing cones*, which intersect in *viewing zones*. These zones represent the location in space where a user will be able to see the two different signals on both eyes. This principle in the 2-views case is illustrated on Figure 43. As on can see, the viewing zone is quite limited and the user has to be at a correct distance from the screen, and this distance has a very limited validity range. Moreover, the position within this viewing distance is very important, since one time over two, the user is badly positioned since his left eye will receive the right video signal and *vice versa*.
To overcome these limitations, an auto-stereoscopic display can send a higher number of views. This is illustrated on Figure 44. As one can see, here the range of the viewing distance is increased. By moving forward or backward, a user will not necessarily see always the same views, but he will see a correct view on each eye. Moreover, the position comfort is increased: a user can move more freely in the viewing zones and still receive the views in the correct order. A good rule of thumb for the positioning is to consider that with N views, the user (within the viewing distance) will be correctly positioned $N-1$ times over N.
However, remember that all displayable views should be stored in one single image before view splitting. As a consequence, increasing the number of views decreases the possible spatial resolutions the what the user sees, since screens are of limited physical resolution.

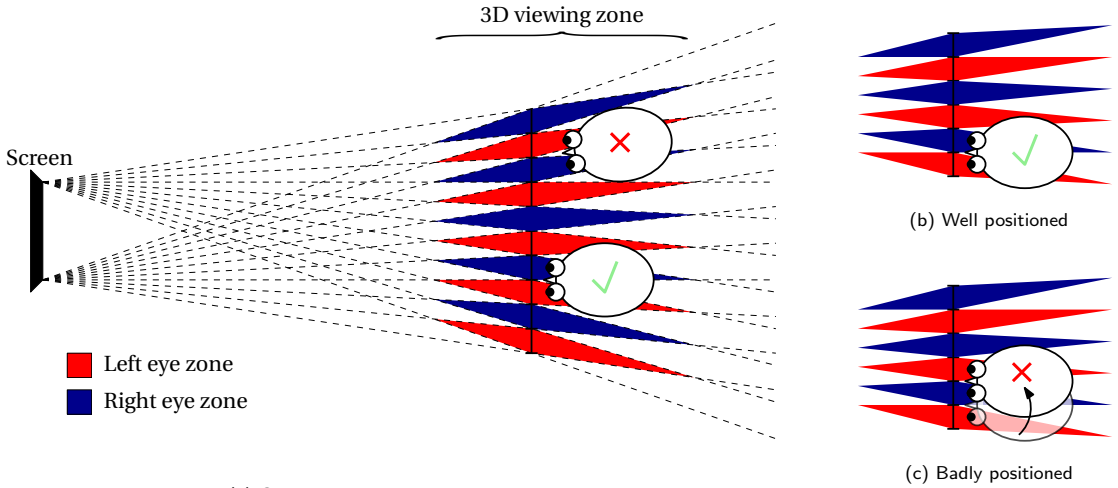


Fig. 43. Limitations of the 2-views auto-stereoscopy

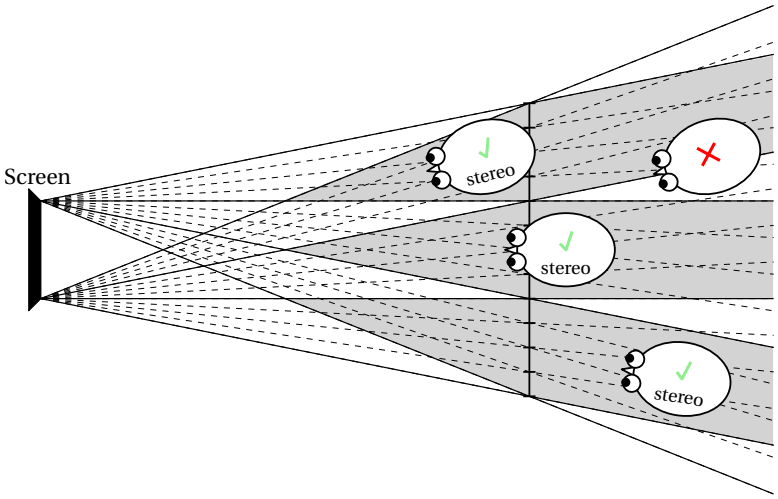


Fig. 44. Increased viewing distance and position with N views

Auto-stereoscopic techniques Several techniques exist to perform the video separation signal on top of the screen. The two principal ones are the *lenticular panel* and the *parallax barrier*. They are illustrated on Figure 45. The lenticular panel is composed of small spherical lenses. Each lens covers several pixels of the LCD screen and deviated the outgoing light in the desired directions so that the user will see different pixels depending on its position. The parallax barrier is composed of a black and opaque mask in front of the screen, pierced with tiny holes letting the screen outgoing light pass through in desired directions. As for the lenticular panel, the barrier's holes are intended to let the user see only a predefined set of LCD pixels. Moreover, due to the masking process, the brightness of the signal the user views may be highly reduced reduced compared to the lens-based system.

11.2 - Implementation of an auto-stereoscopic rendering

If one wishes to implement an application designed to display auto-stereoscopic content with an adequate screen, there are two main issues to solve. First of all, the different views required by the display have to be correctly generated. This requires to know the configuration of the virtual cameras that would

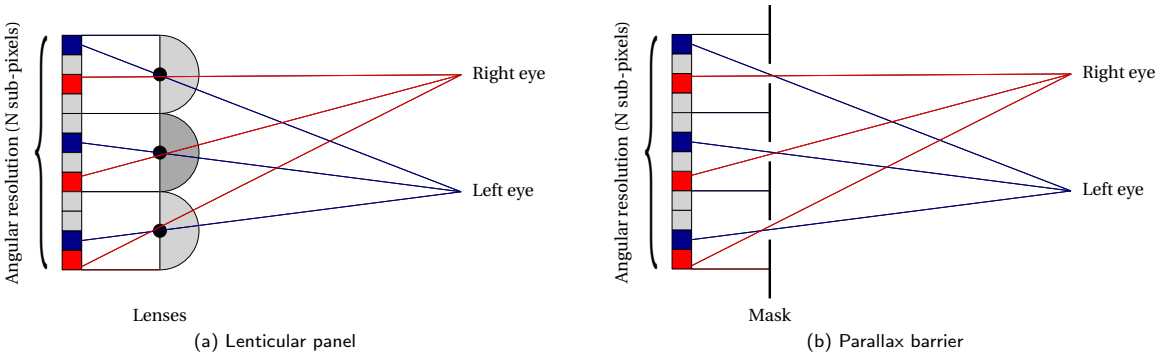


Fig. 45. Examples of auto-stereoscopic techniques

correspond to the different views acquisition. Secondly, once the views are generated, they have to be correctly interleaved in one single image so that one viewed through the auto-stereoscopic layer system, each user's eyes see a different input view.
From now on, we take the example of the design of an OpenGL application targeted to perform auto-stereoscopic rendering, so that both the virtual views generation and interleaving procedures can be described.

11.2.1 - Virtual views generation

In a classical OpenGL rendering engine, without auto-stereoscopic display, the view generation procedure is held in a single infinite display loop. In Listing 1, the method `display()` is called infinitely until the application is stopped. The viewport setup correspond to the determination of the screen zone on which the image will be rendered. The two point we are interested in for the virtual views generation are the setup of the camera pose and projection.

```
@Override
public void display(GLAutoDrawable arg0) {
  // Setup viewport
  ...
  // Setup camera projection
  ...
  // Setup camera pose
  ...
  // Render scene
  ...
}
```

Listing 1. OpenGL typical display loop

Setting the poses We assume from now on that we know some reference camera $\mathcal{C}_r$, which corresponds to the virtual OpenGL camera that would have been used without auto-stereoscopic display. We are interested in knowing how to place the virtual cameras $\mathcal{C}_a^i$ used for auto-stereoscopic rendering. The key point here is to understand that a neighboring camera pair $(\mathcal{C}_a^i, \mathcal{C}_a^{i+1})$ in the virtual world shall represent the users eyes in the real world.
The whole real / virtual correspondence setup is depicted on Figure 46. We consider fronto-parallel virtual cameras, in terms of positioning, as in the multi-view acquisition case. See [MBBB07] for a justification. As a consequence, the information we have to provide to our rendering engine is the relative cameras spread Δ_{cam}. To compute such spread, we match in our model the following information:

- As already stated, the user's eyes are matched to the virtual cameras, thus the distance Δ_{eye} is associated to the desired camera spread Δ_{cam}.

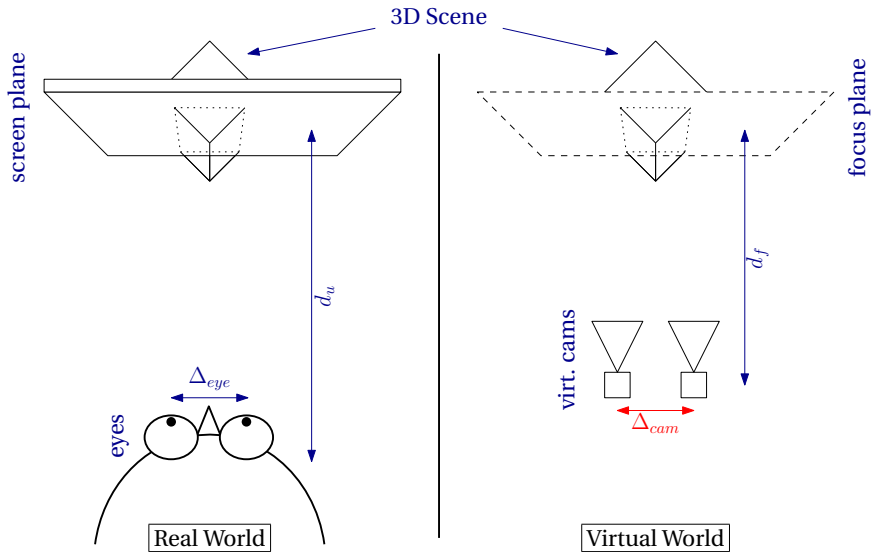


Fig. 46. Relationship between user's eyes and virtual cameras positioning

↬ In auto-stereoscopic display, one has the sensation that some viewed objects are in front of the screen while others are behind. With this statement in mind, we define a virtual and non displayed 3D plane in the virtual space, parallel to the camera's image planes. We call it the *focus plane*. So the screen in the real world is matched to the virtual focus plane, and the distance d_u between the user and the screen is associated to the distance d_f between the virtual cameras and the focus plane.

With such modeling of the virtual cameras positioning, one can easily deduce the camera spread:

$$\Delta_{cam} = \alpha \frac{\Delta_{eye} \cdot d_f}{d_u}, \tag{33}$$

with α being an additional scalar allowing to arbitrarily reduce or strengthen the auto-stereoscopic effect.

Setting the projection Defining the cameras' projection matrices can be performed in several ways in OpenGL. The hard constraint set by the auto-stereoscopic display is that we want the display scene to be rendered in such a way that all image planes can be superimposed, contrary to multi-view acquisitions described in Part II. This is illustrated in Figure 47a. To setup such projection, we must first remind how it can be defined in OpenGL.

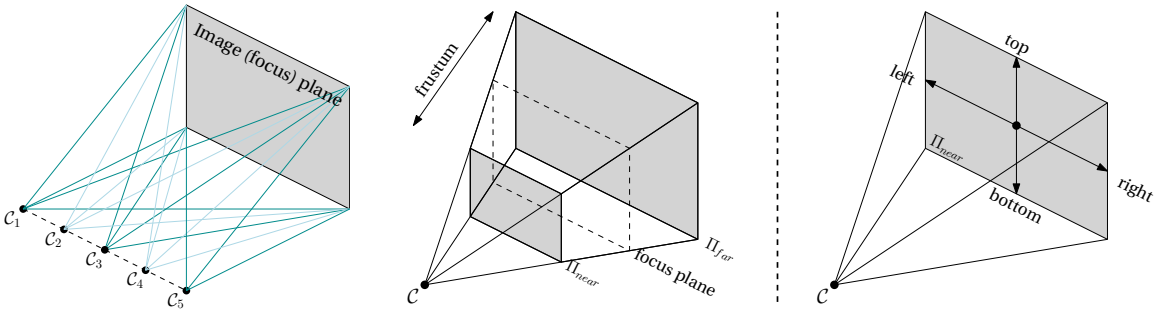
The rendering engine only displays the virtual objects that fall into the *frustum*. It is a truncated pyramid defined by the camera center, the image plane, and two cutting planes: the near plane (Π_{near}) and the far plane (Π_{far}). The focus plane is between them (see Figure 47b, left part). These information are sufficient to define a simple pinhole camera projection.

In OpenGL framework, such view frustum can be defined using the function `glFrustum()`, which takes six input distances parameters: the near and far distances for the cutting planes, and four additional distances defining the near plane borders (see Figure 47b, right part). These are distances expressed in the near plane's space, from the projected optical center.

As one may guess, to define the cameras' projection in our auto-stereoscopic setup, it is sufficient to shift the *left* and *right* parameter distances with the correct amount so that all the focus planes will match.

11.2.2 - Virtual views interleaving

Once a camera has been correctly set, the scene is rendered for the desired point of view. This process is repeated until all necessary points of view have been computed. Then the rendered images are interleaved and displayed onto the auto-stereoscopic screen. We describe in this section how to efficiently store the temporary images on one hand, and how to interleave them correctly for the targeted display.



(a) Non centered projection (b) OpenGL projection parameters

Fig. 47. Projection modeling for auto-stereoscopy

Frame buffer objects The Frame Buffer Object architecture (FBO) is an extension of OpenGL for doing flexible off-screen rendering, including — and that's what we are interested in here — rendering to a texture. It's general purpose is to capture images that would normally be drawn to the screen, to implement a large variety of image filters or post-processing effects. The advantage of FBOs in terms of speed is that they do not suffer from the overhead associated with OpenGL drawing context switching. They are also known to largely outperform other off-screen rendering techniques, such as Pixel Buffers. The classical way to use a FBO for rendering to textures is to create it, attaching a depth buffer to it to perform standard depth tests, and attach the desired number of texture objects. The FBO creation procedure is summarized in Listing 2.
In our context, once a virtual camera has been parameterized, we do not render the scene in the standard frame buffer (which is displayed on the screen), but rather in one of the texture units attached to a FBO. Thus the display loop will resemble to this:

1. Select (bind) the Frame Buffer Object
2. For each desired point of view
   (a) Select the texture unit to draw on in th FBO
   (b) Configure the camera pose and projection
   (c) Render the scene
3. Interleave and display all textures attached to the FBO
4. Unbind the FBO

Notice that it is much more efficient to switch between textures within a FBO than between FBOs themselves. As such, depending on the desired number of views, one may want to maximize the number of textures attached to a FBO (which is limited and hardware dependent).

Shader-based interleaving At this point, we consider that the scene has been rendered for all the wanted views, and stored in the FBO's attached textures. Nothing has been displayed to the screen yet. To perform the interleaving, we use a Fragment Shader. The principle is the following: OpenGL's state is restored so that we can render objects on the screen. A simple quadrilateral, which covers the entire screen, is then displayed. The trick is to tell OpenGL that this quad has multiple textures, which are the textures computed earlier. Then the shader is applied to interleave these multiple textures and provide a final unique texture applied and displayed on the quad — and of course on the screen.
/newpar The only remaining problem is to know the interleaving function. This is an issue in the sense that the interleaving for a given pixel depends from its position on the screen, and that this function is hardware-dependent (with regards to the auto-stereoscopic display used). As such, it is generally very hard to model this function for a given screen. To overcome this problem, we use instead interleaving masks, which can be retrieved by reverse engineering. These masks are defined as additional textures attached to the quad, and as such they are fed to fragment shader program, in a multiply-additive way:

$$\forall \text{ pixel } \mathbf{x},\ \text{shader}(\mathbf{x}) = \sum_{i=1}^{N} \text{texture}(\mathbf{x}) * \text{mask}(\mathbf{x}),\ \text{ with } N \text{ the number of views} \qquad (34)$$

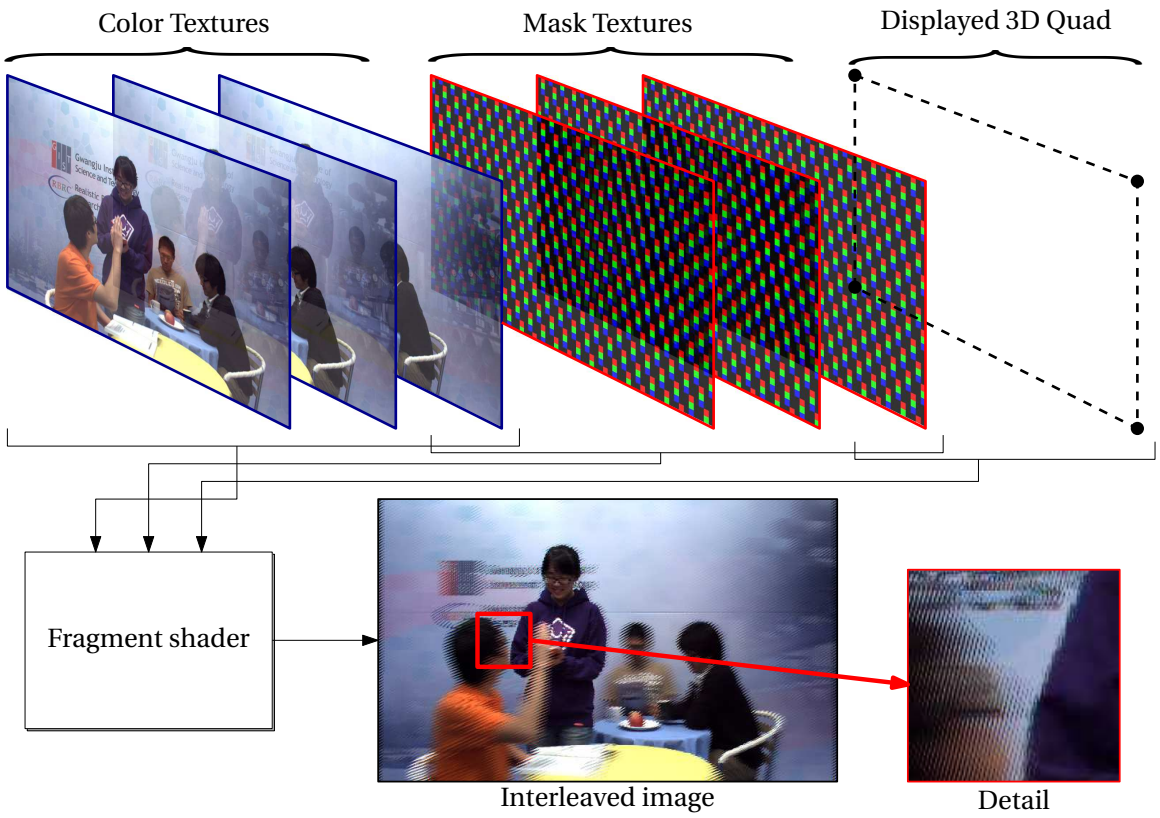


Fig. 48. Fragment shader-based interleaving process

The shader program itself is very simple. A GLSL version of it is given in Listing 3, for a display function using eight views. The whole interleaving procedure is summarized on Figure 48.

Code excerpts

```
public boolean initFBO(int width, int height) {

   // Check if extension is available on current graphic implementation
   if(!gl.isExtensionAvailable("GL_EXT_framebuffer_object")) {
      System.err.println("GL_EXT_framebuffer_object not supported !");
      return false;
   }

   // Generate frame buffer object and store id (idFBO is an int)
   int idtab[] = new int[1];
   gl.glGenFramebuffersEXT(1, idtab, 0);
   idFBO = idtab[0];

   // Bind the FBO to make its context active
   gl.glBindFramebufferEXT(GL.GL_FRAMEBUFFER_EXT, idFBO);

   // Generate render buffer object and save id (idRenderBuffer is an int)
   gl.glGenRenderbuffersEXT(1, idtab, 0);
   idRenderBuffer = idtab[0];

   // Initialize depth storage
   gl.glBindRenderbufferEXT(GL.GL_RENDERBUFFER_EXT, idRenderBuffer);
   gl.glRenderbufferStorageEXT(GL.GL_RENDERBUFFER_EXT,
      GL.GL_DEPTH_COMPONENT, width, height);
   gl.glBindRenderbufferEXT(GL.GL_RENDERBUFFER_EXT, 0);
```

```
26
27    // Attach depth storage to FBO
28    gl.glFramebufferRenderbufferEXT(GL.GL_FRAMEBUFFER_EXT, GL.GL_DEPTH_ATTACHMENT_EXT,
29       GL.GL_RENDERBUFFER_EXT, idRenderBuffer);
30
31    // Here for the example we will attach two textures. Beware that the number of available
32    // textures to be attached is hardware dependent, and proper tests should be run here.
33
34    // Generate textures objects ids
35    // Note: int idTexs[] = new int[2];
36    gl.glGenTextures(2, idTexs, 0);
37
38    // Create texture objects and attach them to the FBO as color buffers
39    for(int i = 0; i < n; i++) {
40      gl.glBindTexture(GL.GL_TEXTURE_2D, idTexs[i]);
41      gl.glTexParameteri(GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MIN_FILTER, GL.GL_NEAREST);
42      gl.glTexParameteri(GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MAG_FILTER, GL.GL_NEAREST);
43      gl.glTexParameterf(GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_S, GL.GL_CLAMP_TO_EDGE);
44      gl.glTexParameterf(GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_T, GL.GL_CLAMP_TO_EDGE);
45      gl.glTexImage2D(GL.GL_TEXTURE_2D, 0, GL.GL_RGB, width, height, 0,
46         GL.GL_RGB, GL.GL_UNSIGNED_BYTE, null);
47      gl.glBindTexture(GL.GL_TEXTURE_2D, 0);
48      gl.glFramebufferTexture2DEXT(GL.GL_FRAMEBUFFER_EXT, GL.GL_COLOR_ATTACHMENT0_EXT+i,
49         GL.GL_TEXTURE_2D, idTexs[i], 0);
50    }
51
52    // Unbind the FBO
53    gl.glBindFramebufferEXT(GL.GL_FRAMEBUFFER_EXT, 0);
54
55    return true;
56 }
```

Listing 2. FBO creation

```
1  uniform sampler2D images[8];
2  uniform sampler2D masks[8];
3
4  void main() {
5     vec4 color0 = texture2D(images[0], gl_TexCoord[0].st);
6     vec4 color1 = texture2D(images[1], gl_TexCoord[0].st);
7     vec4 color2 = texture2D(images[2], gl_TexCoord[0].st);
8     vec4 color3 = texture2D(images[3], gl_TexCoord[0].st);
9     vec4 color4 = texture2D(images[4], gl_TexCoord[0].st);
10    vec4 color5 = texture2D(images[5], gl_TexCoord[0].st);
11    vec4 color6 = texture2D(images[6], gl_TexCoord[0].st);
12    vec4 color7 = texture2D(images[7], gl_TexCoord[0].st);
13    vec4 mask0 = texture2D(masks[0], gl_TexCoord[0].st);
14    vec4 mask1 = texture2D(masks[1], gl_TexCoord[0].st);
15    vec4 mask2 = texture2D(masks[2], gl_TexCoord[0].st);
16    vec4 mask3 = texture2D(masks[3], gl_TexCoord[0].st);
17    vec4 mask4 = texture2D(masks[4], gl_TexCoord[0].st);
18    vec4 mask5 = texture2D(masks[5], gl_TexCoord[0].st);
19    vec4 mask6 = texture2D(masks[6], gl_TexCoord[0].st);
20    vec4 mask7 = texture2D(masks[7], gl_TexCoord[0].st);
21    gl_FragColor = color0 * mask0 + color1 * mask1 + color2 * mask2 + color3 * mask3 +
22                   color4 * mask4 + color5 * mask5 + color6 * mask6 + color7 * mask7;
23 }
```

Listing 3. Fragment shader for auto-stereoscopic interleaving

11.3 - Example: auto-stereoscopic 2D+Z rendering

The auto-stereoscopic rendering engine described in Section 11.2 is generic with regards to the rendered scene. We now describe how to render 2D + Z videos in such framework.

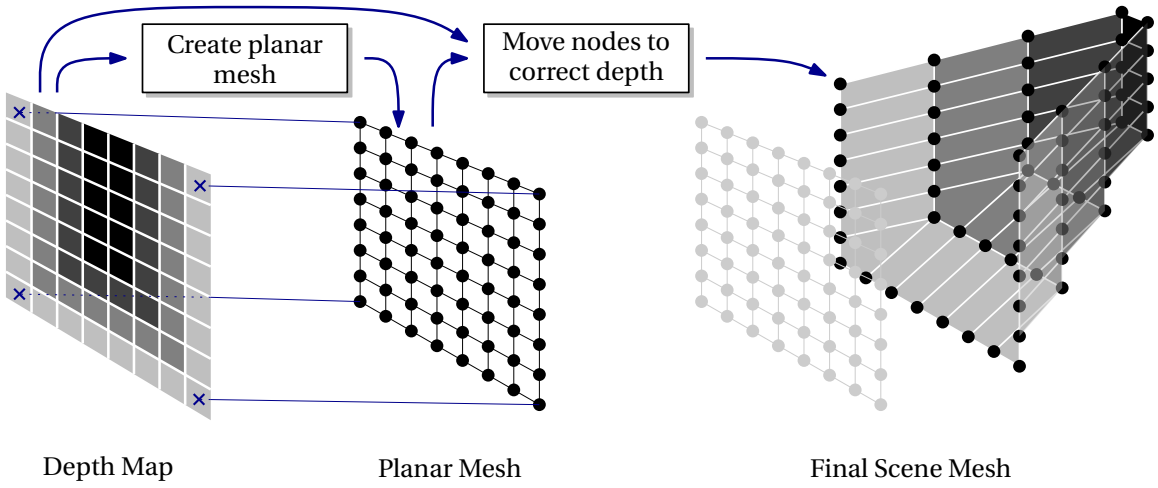


Fig. 49. Construction of a 3D mesh for 2D+Z rendering

11.3.1 - Representing the scene in 3D

2D + Z videos are sequences composed of standard images (the texture information) and depth maps (the 3D information). In our auto-stereoscopic scheme, a 3D scene has to be rendered using OpenGL routines. As a consequence, the input sequences have to be converted into 3D textured meshes. With few additional information, a depth map can be converted into such mesh.

To compute a mesh representing the scene at a given time from a depth map, we first build a regular planar mesh, whose nodes correspond to pixel positions. This mesh is associated to the image plane of the camera. Then, every single node of the mesh is pushed along its line of sight such that its Z coordinate in the camera frame is equal to the value read in the depth map. This mesh is then textured with the original image from the video. This is illustrated in Figure 49.

Here three parameters are required: the focal length f, and the nearest and farthest depth values z_{min} and z_{max}. The only constraint is that the three of them be expressed with the same units, but the unit itself does not matter. These parameters are generally given the the depth maps providers, for instance the DERS or our depth map extraction software *mv2mvd*. The focal length is physically the distance between the camera sensor (CCD or roll film) and the lens' optical center. The two last parameters are simply the mapping parameters used to convert the greyscale depth maps image values to the desired depths.

Following this idea, the function mapping a pixel position $\mathbf{x} = [x,\ y]^\top$ to a 3D point $\mathbf{X} = [X,\ Y,\ Z]^\top$ is the following (w and h are the input video dimensions, d is the value read in the depth map):

$$
\begin{aligned}
&1 \quad - \quad \text{Initialize } \mathbf{X} \text{ to } \bar{\mathbf{X}} \text{ to build the planar mesh:} \\
&\qquad \bar{\mathbf{X}} = \left[x - \tfrac{w}{2},\ y - \tfrac{h}{2},\ z_{min}\right] \\
&2 \quad - \quad \text{Build } \mathbf{X} \text{ by modifying } \bar{\mathbf{X}}: \\
&\qquad \mathbf{X} = \begin{cases} Z &= \bar{Z} + (z_{max} - z_{min}) * d \\ X &= \bar{X} * Z/f \\ Y &= \bar{Y} * Z/f \end{cases}
\end{aligned}
\tag{35}
$$

11.3.2 - Real-time 3D depth mapping

We saw in Section 11.2 that real-time rendering is an important issue in auto-stereoscopic display, and that interleaving operations for instance have to be run on the GPU through a shader to be performed in real time. The problem is basically the same here: 3D mesh generation from depth maps has to be done

very fast in case one has to view a video, let's say at 25Hz, without loosing the real time auto-stereoscopic ability.

The solution for such model generation is once again found in the extensive use of shaders. This time, we do not use a fragment shader performing operations on "pixels", but rather a vertex shader that operates directly on the given 3D points. The other trick used here is to be aware that in depth map-based rendering, the 3D rendered scene is viewed and expressed *in the camera coordinate system*. As a consequence, the 3D planar mesh defined before modifying its nodes' position is always exactly the same for every single image of the video. It follows that this mesh is built once and for all before rendering the first image, and the modification of its nodes is done on the flow in the vertex shader. One has never to modify these points position outside the shader. Then, for all the video it is always the same mesh that is displayed. The vertex shader program modifying the nodes' positions is given in Listing 4. Notice that in this example, the Z coordinates are negative due to OpenGL default coordinate system.

```
uniform float zmin;
uniform float zmax;
uniform float focal;
uniform sampler2D depth[1];

void main() {
   // Map the multi-texture coordinates to classical texture coordinates in [0;1]
   // Multi-texturing is used because of the 2 color / depth textures
   gl_TexCoord[0] = gl_MultiTexCoord0;
   // Read the depth value from the depth texture
   float readDepth = texture2D(depth[0], gl_TexCoord[0].st).x;
   // Inhomogeneous normalization
   gl_Vertex   /= (gl_Vertex.w);
   // Compute the new vertex position
   gl_Vertex.z -= (zmax-zmin) * readDepth;
   gl_Vertex.x *= -(gl_Vertex.z/focal);
   gl_Vertex.y *= -(gl_Vertex.z/focal);
   // Mandatory: Compute the position of the projected vertex in the camera frame,
   // including the intrinsics influence. This is the mandatory output of the vertex shader
   gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

Listing 4. Vertex shader for 2D+Z rendering

11.4 - Further notes

Software This auto-stereoscopic rendering engine associated to the viewing of 2D + Z videos has been successfully implemented in a small Java application named *M3dPlayer Lite*. It runs in ('almost always', see notes below) real-time on a Linux 32bits computer with 2GB of RAM and a Inter®Core™2 Duo 6700 CPU, and a Quadro FX 3500M graphics card. It has been demonstrated with several videos, with our extracted depth maps.

Depth maps quality In such auto-stereoscopic context, it is highly hard to differentiate between our depth maps or DERS' ones. This comes from the fact that auto-stereoscopy tends to generate very local virtual views, thus lowering the depth artifacts influence. Moreover, it appears that false depths in large textureless regions does not impact on perceptual quality of 3D experience. A final concern on depth maps quality is that for auto-stereoscopy purposes, extremely accurate depth contours estimation is not an issue. It may in fact be a problem due to our internal mesh representation, since it is completely connected, leading to texture stretchings at depth contours when not correctly positioned with regards to the screen.

Input videos resolution Another point to keep in mind is that spatial resolution of 3D visualization is driven not by the input data, but by the auto-stereoscopic screen itself, which is much lower that the LCD panel itself. As such it is pointless to feed the application with for instance HD content when the device

itself will output for each eye SD images. This would have for consequence to increase the required texture allocation for both images and depth maps, and the number of 3D points rendered by OpenGL ; this may drop the rendered frames per second number drastically. That is why in the first paragraph of this section we said that rendering is almost always performed in real-time: it is not the case when one provides irrelevant input content.

The Frame Buffer Object trick With the same resolution wonders, it is very important to notice that the resolution of the textures attached to the FBO do not need to be at full resolution, even if they are stretched by OpenGL to fit the final quadrilateral dimensions (in a very good manner by the way). In our application, we use square texture units of size 1024×1024 pixels. Increasing this resolution may also make the FPS fall since the memory amount on the graphics card is much more limited. In our case we have 256MB of such memory. However, the mask textures shall them be at full resolution since they define pixel-wise interpolation coefficients and must not be corrupted by images interpolations.

Appendixes

A - The depth map model

Depth maps can be considered as 3D information added to images. They are generally encoded as grayscale images, to which one generally associates corresponding min and max depth values for scaling purposes: black matches to a minmal depth while white matches a maximal depth. Depths in between are linearly interpolated. Another formulation, used in Z-Buffering within graphics card, is non linear between the min and the max, so that more numerical precision be allocation to near objects.

Each pixel in the depth maps describes the depth of the corresponding pixel in the image, *relative to the camera coordinate system*. In the classical $(X, Y, Z) \in \mathbb{R}^3$ camera space, the stored value corresponds to the Z coordinate of the point (see Figure 50).

Given an image pixel $p = (x, y)$ and a corresponding depth value Z_P , the complete 3D coordinates are retrieved using the camera focal length f:

$$\begin{aligned} X_P &= x \cdot \frac{Z_P}{f} \\ Y_P &= y \cdot \frac{Z_P}{f} \end{aligned} \tag{36}$$

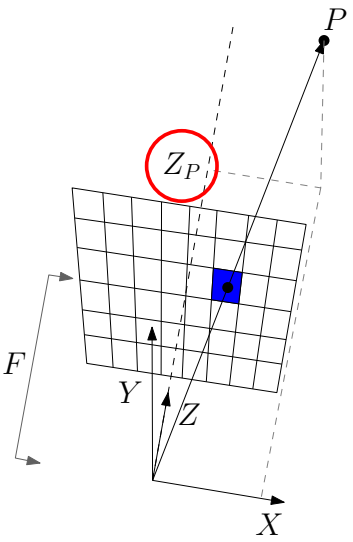


Fig. 50. Depth map principle

B - Rotations and their representations

Several representations exists to model rotations in 3D space. The one used in the pihole camera model, and in projection equations, is the 3×3 matrix formulation. However, hard constraints are set on such matrix formulation: they are orthogonal and their determinant is equal to 1.

$$\mathbf{R}^\top = \mathbf{R}^{-1} \text{ and } \det(\mathbf{R}) = 1 \tag{37}$$

These constraints are attibutable to the dimension of the rotations in 3D space, which is 3.

Such constraints are hard ot maintain in several rotation-related computation, such as rotations interpolations or derivations in non linear optimization frameworks. As such, other equivalent but better dimensionned formulations are preferred in these cases. We present here two of them: the axis-angle formulation, and the quaternion formulation.

B.1 - Axis-angle formulation

The axis-angle representation, also known as the exponential coordinates of a rotation, parameterizes the rotation by a unit oriented 3-vector ϕ giving the rotation direction, and a scalar θ representing the angle, or the amount, of the rotation (with the right hand grip rule). This representation comes from Euler's rotation theorem [Eul76].

We use this formulation as a simple way to compute rotations interpolations in 3D space.

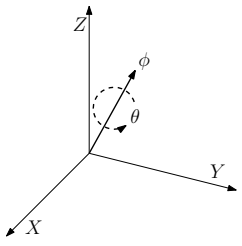


Fig. 51. Axis-angle rotation formulation

B.1.1 - From matrix to axis-angle representation

The transformation between a rotation matrix to an axis-angle representation is performed in two successive steps. First, the rotation angle is computed:

$$\theta = \arccos\left(\frac{\text{trace}(\mathbf{R}) - 1}{2}\right) \tag{38}$$

Then, the angle is used to determine the normalized axis of the rotation:

$$\phi = \frac{1}{2\sin(\theta)}\begin{bmatrix}\mathbf{R}_{3,2} - \mathbf{R}_{2,3}\\ \mathbf{R}_{1,3} - \mathbf{R}_{3,1}\\ \mathbf{R}_{2,1} - \mathbf{R}_{1,2}\end{bmatrix} \tag{39}$$

B.1.2 - From axis-angle to matrix representation

The transformation between an axis-angle rotation to a rotation matrix is drived by the exponential map transformation, leading to the well known Rofrigues' rotation formula:

$$\mathbf{R} = \mathbf{I} + [\phi]_\times \sin(\theta) + [\phi]_\times^2 (1 - \cos(\theta)), \tag{40}$$

with $[\cdot]_\times$ beeing the operator giving the antisymmetric matrix equivalent to the cross-product.

B.2 - Quaternion formulation

Quaternions can be seen as an extension of complex numbers, mainly used in mechanics in three-dimentional space [DFM07]. To make it very quick[6] if complex numbers have a real and an imaginary part, then quaternions have a real part and a three-dimentional imaginary part: $\mathbf{q} = q_0 + q_1\,\mathbf{i} + q_2\,\mathbf{j} + q_3\,\mathbf{k} = (q_0, [q_1, q_2, q_3]^\top)$.

Given a rotation represented with the axis-angle (θ, ϕ), the equivalent rotation quaternion has the following form:

$$\mathbf{q} = \left(\cos\left(\frac{\theta}{2}\right), \phi\,\sin\left(\frac{\theta}{2}\right)\right) \tag{41}$$

One speaks of unit quaternions, since their euclidean norm equals 1.

B.2.1 - Rotating a point or vector in 3-space with quaternions

One of the reason why quaternions are very usefull is that rotating a point or a vector can be performed directly with quaternion multiplications. Let $\mathbf{X}$ be a 3D point. It can be represented by a quaternion with a null real part and its coordinates assigned to the imaginary part: $\mathbf{q_x} = \left(0, [X, Y, Z]^\top\right)$. Rotating $\mathbf{X}$ with the rotation described in a quaternion $\mathbf{q}$ is done by multiplying the corresponding imaginary quaternion by $\mathbf{q}$ and its conjugate $\mathbf{q}^*$ in the Hamilton product sense:

$$\mathbf{X}' = \mathbf{R}\mathbf{X} = \mathbf{q}\,\mathbf{q_x}\,\mathbf{q}^* \tag{42}$$

It can of course be easily verified that the real part of $\mathbf{X}'$ is equal to zero.

[6] If you don't know anything about quaternions, Google them, you'll find their main properties needed for further understanding (conjugation, norm, multiplication, etc.)

B.2.2 - Minimal representation

One can also notice that such rotation quaternions can be represented only with their imaginary part, leading to a 3-vector space representation corresponding to the exact dimension of a rotation, since the real part can be deduced from the imaginary part, using its norm:

$$\left\|[q_1,q_2,q_3]^\top\right\|^2 = \sin^2\left(\frac{\theta}{2}\right) \;\Rightarrow\; q_0 = \cos\left(\frac{\theta}{2}\right) = \sqrt{1-\left\|[q_1,q_2,q_3]^\top\right\|^2} \tag{43}$$

These properties are particularly interesting in our context, more precisely when we need to derivate projection equations in non-linear optimization frameworks, where the modified rotation values need to maintain the rotations properties (see for instance Section C).

B.2.3 - From quaternion to axis-angle representation

The transformation from an axis-angle representation to a quaternion beeing straightforward (only a cosine and a sine need to be computed), we just remind here the reverse transformation (which is quite strightforward too).

First, the rotation angle θ can be deduced from the real part of the quaternion, and then the unit vector ϕ is derived by a scalar division of the imaginary part:

$$\theta = 2\,\arccos(q_0) \quad \text{and} \quad \phi = \frac{[q_1,q_2,q_3]^\top}{\sin(\frac{\theta}{2})} \tag{44}$$

B.2.4 - From quaternion to matrix representation

The matrix formulation of the unit rotation quaternion $\mathbf{q} = (q_0, [q_1,q_2,q_3]^\top)$ is the following:

$$\mathbf{R} = \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2q_1q_2-2q_0q_3 & 2q_1q_3+2q_0q_2 \\ 2q_1q_2+2q_0q_3 & q_0^2-q_1^2+q_2^2-q_3^2 & 2q_2q_3-2q_0q_1 \\ 2q_1q_3-2q_0q_2 & 2q_2q_3+2q_0q_1 & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix} \tag{45}$$

B.2.5 - From matrix to quaternion representation

Finding a quaternion equivalent to a rotation matrix $\mathbf{R}$ can be numerically unstable when the trace of $\mathbf{R}$ is close to zero. We describe here a robust method to find $\mathbf{R}$ given $\mathbf{q}$.
First, let $\mathbf{R}_{a,a}$ be the diagonal element with the largest absolute value. We compute the temporary value $r = \sqrt{1+\mathbf{R}_{a,a}-\mathbf{R}_{b,b}-\mathbf{R}_{c,c}}$, with abc being an even permutation of xyz (*i.e.* xyz, zxy of yzx). The quaternion indexes $1,2,3$ are mapped in the notation to x,y,z. The quaternion $\mathbf{q}$ may now be written:

$$\begin{cases} q_0 &= (\mathbf{R}_{c,b}-\mathbf{R}_{b,c})/2r \\ q_a &= r/2 \\ q_b &= (\mathbf{R}_{a,b}+\mathbf{R}_{b,a})/2r \\ q_c &= (\mathbf{R}_{c,a}+\mathbf{R}_{a,c})/2r \end{cases} \tag{46}$$

C - Derivating projection equations

We provide in this section the details of the partial derivative of the projection equations with regards to rotations, translations, structure (*i.e.* 3D points) and camera intrinsics. These derivatives are used to build the Jacobian matrices in Levenberg-Marquardt optimizations, such as resection or bundle adjustment.

C.1 - Projection equations

We consider the standard projection relationship $\mathbf{x} \sim \mathbf{K}[\mathbf{R}|\mathbf{t}]\mathbf{X}$. We want to express the coordinates x and y from $\mathbf{x}$, using the quaternion representation for the rotation $\mathbf{R}$, namely $\mathbf{q} = (r_a, [r_x, r_y, r_z]^\top)$, with $r_a = \sqrt{1 - r_x^2 - r_y^2 - r_z^2}$.
Let also (f_x, f_y, u_0, v_0) be the camera intrinsic parameters, (X, Y, Z) the coordinates of the original 3D point $\mathbf{X}$, and (t_x, t_y, t_z) the translation coordinates.

We define the following temporary variable $\mathbf{Q_X} = [Q_X, Q_Y, Q_Z]^\top$, corresponding to the coordinates of $\mathbf{X}$ expressed in the camera coordinate system ($\mathbf{Q_X} = \mathbf{R}\mathbf{X} + \mathbf{t}$):

$$\mathbf{Q_X} = \begin{cases} Q_X &= X\left(1 - 2r_y^2 - 2r_z^2\right) + 2\,Y\left(r_x r_y - r_a r_z\right) + 2\,Z\left(r_a r_y + r_x r_z\right) + t_x \\ Q_Y &= Y\left(1 - 2r_x^2 - 2r_z^2\right) + 2\,X\left(r_a r_z + r_x r_y\right) + 2\,Z\left(r_y r_z - r_a r_x\right) + t_y \\ Q_Z &= Z\left(1 - 2r_x^2 - 2r_y^2\right) + 2\,X\left(r_x r_z - r_a r_y\right) + 2\,Y\left(r_a r_x + r_y r_z\right) + t_z \end{cases} \tag{47}$$

We can now write the projection equations, *i.e.* the projected 2D coordinates of $\mathbf{x}$ regarding structure, motion and camera parameters:

$$\begin{aligned} x &: (\mathbf{K}, \mathbf{q}, \mathbf{t}, \mathbf{X}) \mapsto \frac{f_x Q_X}{Q_Z} + u_0 \\ y &: (\mathbf{K}, \mathbf{q}, \mathbf{t}, \mathbf{X}) \mapsto \frac{f_y Q_Y}{Q_Z} + v_0 \end{aligned} \tag{48}$$

C.2 - Structure partial derivatives

In this section, we derive the projection equations 48 with regards to the structure parameters, namely the coordinates of the 3D point $\mathbf{X}$.

$$\begin{aligned}
\frac{\partial x}{\partial X} &= \frac{f_x}{Q_Z}\left(1 - 2r_y^2 - 2r_z^2\right) - 2\,\frac{f_x Q_x}{Q_Z^2}\left(r_x r_z - r_a r_y\right) \\
\frac{\partial x}{\partial Y} &= 2\,\frac{f_x}{Q_Z}\left(r_x r_y - r_a r_z\right) - 2\,\frac{f_x Q_x}{Q_Z^2}\left(r_a r_x + r_y r_z\right) \\
\frac{\partial x}{\partial Z} &= 2\,\frac{f_x}{Q_Z}\left(r_a r_y + r_x r_z\right) - \frac{f_x Q_x}{Q_Z^2}\left(1 - 2r_x^2 - 2r_y^2\right) \\
\frac{\partial y}{\partial X} &= 2\,\frac{f_y}{Q_Z}\left(r_a r_z + r_x r_y\right) - 2\,\frac{f_y Q_y}{Q_Z^2}\left(r_x r_z - r_a r_y\right) \\
\frac{\partial y}{\partial Y} &= \frac{f_y}{Q_Z}\left(1 - 2r_x^2 - 2r_z^2\right) - 2\,\frac{f_y Q_y}{Q_Z^2}\left(r_a r_x + r_y r_z\right) \\
\frac{\partial y}{\partial Z} &= 2\,\frac{f_y}{Q_Z}\left(r_y r_z - r_a r_x\right) - \frac{f_y Q_y}{Q_Z^2}\left(1 - 2r_x^2 - 2r_y^2\right)
\end{aligned}$$

C.3 - Motion partial derivatives

In this section, we derive the projection equations 48 with regards to the six camera pose parameters, namely the three translation ones $[t_x, t_y, t_z]^\top$ and the three rotation ones $[r_x, r_y, r_z]^\top$.

$$\frac{\partial x}{\partial r_x} = \frac{2f_x}{Q_Z}\left(Y\left(r_y + \frac{r_x r_z}{r_a}\right) + Z\left(r_z - \frac{r_x r_y}{r_a}\right) - \frac{Q_X\left(X\left(r_z + \frac{r_x r_y}{r_a}\right) + Y\left(r_a - \frac{r_x^2}{r_a}\right) - 2Zr_x\right)}{Q_Z}\right)$$

$$\frac{\partial x}{\partial r_y} = \frac{2f_x}{Q_Z}\left(Y\left(r_x + \frac{r_y r_z}{r_a}\right) + Z\left(r_a - \frac{r_y^2}{r_a}\right) - 2Xr_y - \frac{Q_X\left(X\left(\frac{r_y^2}{r_a} - r_a\right) + Y\left(r_z - \frac{r_x r_y}{r_a}\right) - 2Zr_y\right)}{Q_Z}\right)$$

$$\frac{\partial x}{\partial r_z} = \frac{2f_x}{Q_Z}\left(Y\left(\frac{r_z^2}{r_a} - r_a\right) + Z\left(r_x - \frac{r_y r_z}{r_a}\right) - 2Xr_z - \frac{Q_X\left(X\left(r_x + \frac{r_y r_z}{r_a}\right) + Y\left(r_y - \frac{r_x r_z}{r_a}\right)\right)}{Q_Z}\right)$$

$$\frac{\partial y}{\partial r_x} = \frac{2f_y}{Q_Z}\left(X\left(r_y - \frac{r_x r_z}{r_a}\right) + Z\left(\frac{r_x^2}{r_a} - r_a\right) - 2Yr_x - \frac{Q_Y\left(X\left(r_z + \frac{r_x r_y}{r_a}\right) + Y\left(r_a - \frac{r_x^2}{r_a}\right) - 2Zr_x\right)}{Q_Z}\right)$$

$$\frac{\partial y}{\partial r_y} = \frac{2f_y}{Q_Z}\left(X\left(r_x - \frac{r_y r_z}{r_a}\right) + Z\left(r_z + \frac{r_x r_y}{r_a}\right) - \frac{Q_Y\left(X\left(\frac{r_y^2}{r_a} - r_a\right) + Y\left(r_z - \frac{r_x r_y}{r_a}\right) - 2Zr_y\right)}{Q_Z}\right)$$

$$\frac{\partial y}{\partial r_z} = \frac{2f_y}{Q_Z}\left(X\left(r_a - \frac{r_x^2}{r_a}\right) + Z\left(r_y + \frac{r_x r_z}{r_a}\right) - 2Yr_z - \frac{Q_Y\left(X\left(r_x + \frac{r_y r_z}{r_a}\right) + Y\left(r_y - \frac{r_x r_z}{r_a}\right)\right)}{Q_Z}\right)$$

$$\begin{array}{lll} \frac{\partial x}{\partial t_x} = \frac{f_x}{Q_Z} & \frac{\partial x}{\partial t_y} = 0 & \frac{\partial x}{\partial t_z} = -\frac{f_x Q_X}{Q_Z^2} \\ \frac{\partial y}{\partial t_x} = 0 & \frac{\partial y}{\partial t_y} = \frac{f_y}{Q_Z} & \frac{\partial y}{\partial t_z} = -\frac{f_y Q_Y}{Q_Z^2} \end{array}$$

C.4 - Intrinsics partial derivatives

In this section, we derive the projection equations 48 with regards to the four camera intrinsic parameters, namely (f_x, f_y, u_0, v_0).

$$\begin{array}{llll} \frac{\partial x}{\partial f_x} = \frac{Q_X}{Q_Z} & \frac{\partial x}{\partial f_y} = 0 & \frac{\partial x}{\partial u_0} = 1 & \frac{\partial x}{\partial v_0} = 0 \\ \frac{\partial y}{\partial f_x} = 0 & \frac{\partial y}{\partial f_y} = \frac{Q_Y}{Q_Z} & \frac{\partial y}{\partial u_0} = 0 & \frac{\partial y}{\partial v_0} = 1 \end{array}$$

References

[Bal05] Raphaèle Balter. *Construction d'un maillage 3D évolutif et scalable pour le codage vidéo*. Informatique, Université de Rennes 1, Campus de Beaulieu, Rennes, France, May 2005.

[BSe05] OpenGL Architecture Review Board, D. Shreiner, and et al. *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 2*. Addison Wesley, 2005.

[BSL+07] Simon Baker, Daniel Scharstein, J.P. Lewis, Stefan Roth, Michael J. Black, and Richard Szeliski. A database and evaluation methodology for optical flow. In *Proceedings of the IEEE 11th International Conference on Computer Vision (ICCV'07)*, Rio de Janeiro, Brazil, October 2007.

[BTG06] Herbert Bay, Tinne Tuytelaars, and Luc J. Van Gool. Surf: Speeded up robust features. In Ales Leonardis, Horst Bischof, and Axel Pinz, editors, *Proceedings of the European Conference on Computer Vision (ECCV'06)*, volume 3951 of *Lecture Notes in Computer Science*, pages 404–417. Springer, 2006.

[CM02] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 24(5):603–619, 2002.

[DFM07] Leo Dorst, Daniel Fontijne, and Stephen Mann. *Geometric Algebra for Computer Science: An Object-Oriented Approach to Geometry*. Morgan-Kaufmann Publishers, 2007.

[Eul76] Leonhard Euler. Formulae generales pro translatione quacunque corporum rigidorum. *Novi Commentarii academiae scientarum Petropolitanae*, 20:189–207, 1776. Presented to the St. Petersburg Academy on October 9, 1775.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, June 1981.

[FI08] Andrea Fusiello and Luca Irsara. Quasi-euclidean uncalibrated epipolar rectification. In *Proceedings of theInternational Conference on Pattern Recognition (ICPR'08)*, pages 1–4, 2008.

[Gal02] Franck Galpin. *Représentation 3D de séquences vidéo; Schéma d'extraction automatique d'un flux de modèles 3D, applications à la compression et à la réalité virtuelle*. Informatique, Université de Rennes 1, Campus de Beaulieu, Rennes, France, January 2002.

[Har99] Richard Hartley. Theory and practice of projective rectification. *International Journal of Computer Vision (IJCV)*, 35(2):115–127, 1999.

[HS81] Berthold K.P. Horn and Brian G. Schunk. Determing optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[HS88] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 147–151. The Plessey Company plc., 1988.

[HS97] Richard Hartley and Peter Sturm. Triangulation. *Computer Vision and Image Understanding*, 68(2):146–157, November 1997.

[Hub81] Peter J. Huber. *Robust Statistics*. John Wiley and sons, 1981.

[HZ04] Richard I. Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, June 2004.

[Jon] Rob Jones. OpenGL FrameBuffer Object. http://www.gamedev.net/reference/programming/features/fbo1/.

[KSK06] Andreas Klaus, Mario Sormann, and Konrad Karner. Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. In *Proceedings of the 18th International Conference on Pattern Recognition (ICPR'06)*, pages III: 15–18, 2006.

[KZ01] Vladimir Kolmogorov and Ramin Zabih. Computing visual correspondence with occlusions via graph cuts. In *Proceedings of the International Conference on Computer Vision (ICCV'01)*, pages II: 508–515, 2001.

[LA09] Manolis I. A. Lourakis and Antonis A. Argyros. Sba: A software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software (TOMS)*, 36(1):1–30, 2009.

[LKJH10] Eun-Kyung Lee, Yun-Suk Kang, Jae-Il Jung, and Yo-Shun Ho. 3-d video generation using multi-depth camera system. ISO/IEC JTC1/SC29/WG11 Coding of Moving Pictures And Audio, MPEG2010 m17225, Gwangju Institute of Science and Technology (GIST), Kyoto, Japan, January 2010. Proposal.

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[MBBB07] Bruno Mercier, Kévin Boulanger, Christian Bouville, and Kadi Bouatouch. Multiview Autostereoscopic Displays. Research report, INRIA Rennes Bretagne Atlantique, Rennes, France, 2007.

[PBB+06] Nils Papenberg, Andrés Bruhn, Thomas Brox, Stephan Didas, and Joachim Weickert. Highly accurate optic flow computation with theoretically justified warping. *International Journal of Computer Vision (IJCV)*, 67(2):141–158, April 2006.

[PKG99] Marc Pollefeys, Reinhard Koch, and Luc Van Gool. A simple and efficient rectification method for general motion. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'99)*, pages 496–501, 1999.

[Pol04] Marc Pollefeys. Visual 3D modeling from images. In *Proceedings of the Vision, Modeling, and Visualization Conference (VMV'04)*, page 3, Stanford, California, USA, 2004.

[Ros06] Randi J. Rost. *OpenGL(R) Shading Language (2nd Edition)*. Addison-Wesley Professional, January 2006.

[SS02] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1-3):7–42, April 2002.

[SSS08] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Modeling the world from internet photo collections. *International Journal of Computer Vision*, 80(2):189–210, November 2008.

[ST94] Jianbo Shi and Carlo Tomasi. Good features to track. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, Seattle, June 1994.

[SZS03] Jian Sun, Nan-Ning Zheng, and Heung-Yeung Shum. Stereo matching using belief propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 25(7):787–800, 2003.

[TK91] Carlo Tomasi and Takeo Kanade. Shape and motion from image streams: A factorization method part 3 - detection and tracking of point features. Technical report, CMU School of Computer Science, 1991.

[Tro09] Werner Trobin. *Local, semi-global and global optimization for motion estimation*. PhD thesis, Graz University of Technology, December 2009.

[WTP+09] Manuel Werlberger, Werner Trobin, Thomas Pock, Andreas Wedel, Daniel Cremers, and Horst Bischof. Anisotropic huber-L1 optical flow. In *Proceedings of the British Machine Vision Conference (BMVC'09)*, London, UK, 2009. British Machine Vision Association.

[WZ08] Zeng-Fu Wang and Zhi-Gang Zheng. A region based stereo matching algorithm using cooperative optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'08)*, pages 1–8, 2008.

[YWY+06] Qingxiong Yang, Liang Wang, Ruigang Yang, Shengnan Wang, Miao Liao, and David Nistér. Real-time global stereo matching using hierarchical belief propagation. In *Proceedings of the British Machine Vision Conference (BMVC'06)*, pages 989–998. British Machine Vision Association, 2006.

[ZPB07] Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime TV-L1 optical flow. In *Proceedings of the DAGM-Symposium*, pages 214–223, 2007.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803